UNIVERSITY OF OSLO
Department of Informatics

Creating Flexible Heterogeneous Cloud Environments

Eirik T. Vada

Network and System Administration

Oslo University College

June 11, 2012

Creating Flexible Heterogeneous Cloud Environments

Eirik T. Vada

Network and System Administration
Oslo University College

June 11, 2012

Abstract

Today, private clouds are gaining popularity among businesses due to privacy issues and lack of control in public clouds. Most private clouds still have unique APIs which prevents users and businesses to move across different cloud platforms. Combining different private cloud platforms within a production environment may be advantageous, but is not possible with todays system without reimplementing the business logic.

The goal of this thesis is to explore and investigate this problem and create a prototype tool which makes it possible to move virtual machines across heterogeneous cloud environments. The main reason for creating such a tool, is to gain flexibility in cloud and virtualization platform choices and to prevent vendor lock-in.

The working implementation in this thesis shows that the prototype succeeds in creating a total heterogeneous environment with no dependencies, such as shared storage.

Throughout the thesis the performance was measured and analyzed to give an indication on how the expected virtual machine behavior would be. Further efficiency theories were addressed to pursue the thought of bringing different private clouds into separate environments in a coherent manner.

Acknowledgements

My deepest gratitude goes to my supervisor, Ismail Hassan. His level of knowledge and guidance has been a major inspirational factor, and the numerous discussions that have led to several new and improved ideas to bring this thesis forward. Thank you also for giving me the resources that were necessary to complete the task.

Secondly, a special thanks goes to Kyrre M. Begnum for the way he has inspired my curiosity towards virtualization and cloud computing. His lectures and discussions has been a great inspiration to do further research and work in this field for years to come.

There are although certain people that genuinely needs my gratefulness. My Dad. You have been there for me since day one, and inspired me to continue to work hard towards this final master's degree. Without you behind me, pushing me to fulfill this dream, it would never have happened. My Mom. You're such a great person who have been there for me and supporting me from the start. You know I love you, I really miss you, and I will never forget you.

The next person is my fellow student and best friend, Espen, who has been so helpful throughout these 5 years. I could never have done this without your help, guidance and personal/social impact. You have been a true source of motivation and inspiration, which I am so thankful for.

Last, but not least, my deepest love goes to my family, my girlfriend and daughter, who has stood beside me for 5 amazing years. Your continuous patience and support has been so inspiring and encouraging to fulfill this dream. I owe you both so much.

Thank you!

Oslo, May 2012
Eirik T. Vada

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Network services today are usually being run in what we like to call "the Cloud". Although this modern word, it actually dates far back to the 1960's when virtualization became a new trend in terms of partitioning the large mainframe hardware in a more efficient manner. As in recent years virtualization has developed to become a great way for system administrators to abstract themselves from the large physical network topologies. Since they are now able to create virtualized networks, running on one or more physical nodes, it will create huge benefits when configuring and maintaining the running system.

Looking more at the overall achievements by virtualizing services into fewer physical hosts, we can see a huge potential in reducing the infrastructure hardware itself. Meaning that, by reducing old and often redundant physical servers it creates great cost-efficiency.

When people saw the potential of reducing hardware and move to more virtualized systems, it became a natural attraction when the huge companies, like *Google*, *Amazon* and *Rackspace*, presented what we now know as "the Cloud". We can loosely think of Cloud as a virtualized platform hosted by some company which gives smaller companies the ability to deploy and run their whole infrastructure at their "place". If we think of it like this the virtualization aspect talked about earlier, gives the cost-savings in physical running infrastructure a new meaning. Companies will now have the opportunity to run all their infrastructure in another place.

Deploying a whole infrastructure out to "somebody else", gives certain concerns with regards to security. Administrators are not in that much control over the system as before, and this is a major drawback in terms of configuring and handling the system. Private clouds generates the possibility for companies to deploy their own cloud using their own infrastructure, which brings a lot of flexibility for the administrators. Since the cloud will be running within the organization, the administrators will also be in full control over the system as it is.

1.1.1 Brief background

In 1996 Hugh Dingle published a book called "Migration: the biology of life on the move" [1], in which he states:

"The first characteristic of migrants is persistent movement. This activity carries the migrant beyond its original habitat where it obtained resources to a new one in which it also gathers resources; there may, in fact, be new and different resources gleaned at the destination, as well as ones similar to those at the departure point, although this isn't necessarily so."

If we take a look at migration of a virtual machine, the underlying hardware, on which the minimum two physical servers runs, is not that trivial, as the new environment can for example have more RAM or a faster CPU than the other.

Live Migration is a way to migrate virtual machines between physical nodes while the virtual machines are still running without interrupting any running processes. This will have great benefits during maintenance, since the administrators can for example migrate all instances to a given physical node while performing necessary maintenance on the other, and then migrate all VM's back when the job is done without any downtime.

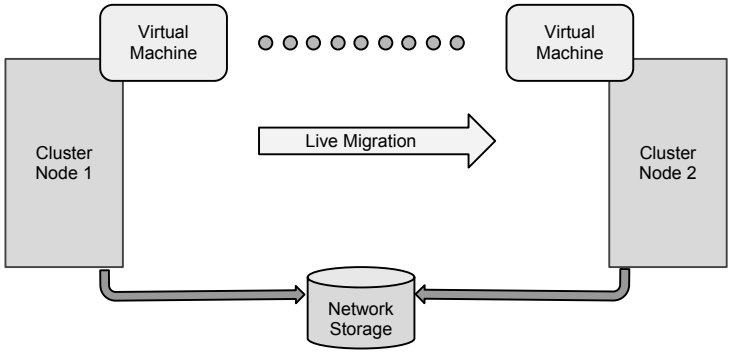

Figure 1.1: Migration of a virtual machine

As the illustration above shows, to be able to migrate a VM from *Cluster Node 1* to *Cluster Node 2* both physical nodes needs a direct connection to a *Network Storage*. As the migration process takes place, the virtual machine image, which is kept on the network storage, remains unaltered during the migration.

1.1.2 Today's problem

Creating multiple and flexible infrastructures, within one or several companies often creates opportunities for easier maintenance, isolation of services and flexibility. These are scenarios that tends to be more and more popular in todays society [2]. Let us now look more into what type of clusters the above ones could be. Because in many cases infrastructures are not running the same hypervisors. Meaning that in many companies you want different hypervisors doing different jobs.

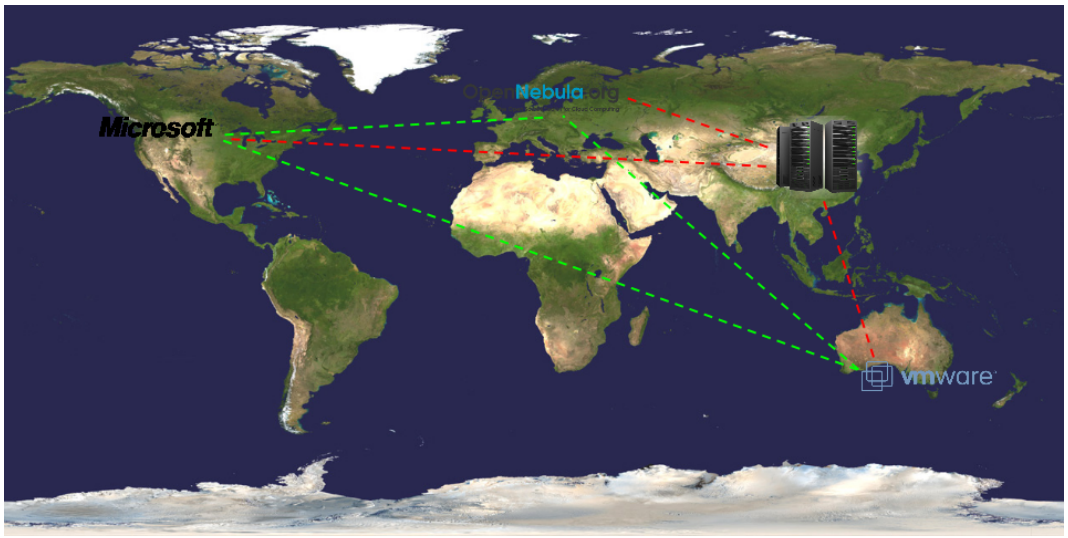

Figure 1.2: Three clouds/infrastructures running at different geographical locations with shared storage

Taking a look at the above picture as an example. One company may run a data-center/cloud in USA, which in this case is Microsoft Hyper-V. Then the company also runs an OpenNebula cloud in Norway and a VMware cloud in Australia. All of them with a direct connection to a shared storage, which in this case is located in China. So how can we live migrate virtual machines across these different hypervisors? The following paper "Heterogeneous live migration of virtual machines" [3] comes up with a solution to the above problem. As the paper states:

"Based on the study of heterogeneity of different VM abstractions and migration algorithms, we designed a common migration framework that provides general abstraction of VMs and migration protocols. We have also implemented a working prototype that supports the live migration of VMs between Xen VMM and KVM."

As the statement mentions, they have made the migration possible between Xen and KVM. In this case it would also be quite simple to add a feature to support Hyper-V as well. Which in that case will solve the example picture talked about above. This kind of solution prevents what is known as "vendor lock-in". Which means that we can now host different types of hypervisors at different locations, and still be able to live migrate instances between them.

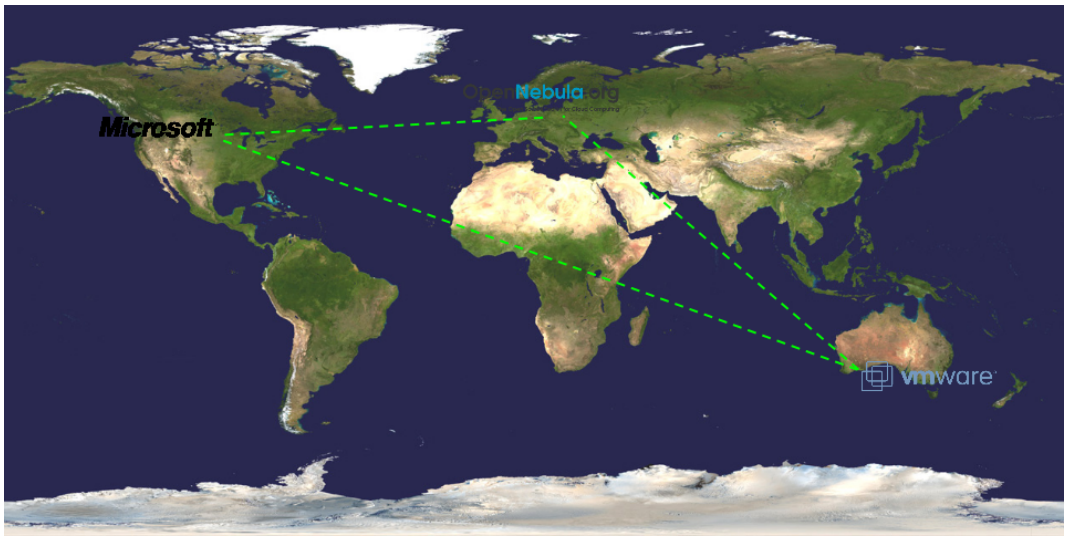

Figure 1.3: Three clouds/infrastructures running at different geographical locations without shared storage

But what if it did not exist any shared storage, meaning that none of the infrastructures above will be "connected" in any way. First of all, the shared storage mentioned earlier will give somewhat a single-point-of-failure. It will most likely be, for example, a redundant SAN, meaning that it will probably handle most failures, but if it goes down, the whole connectivity between all the infrastructures will be terminated.

Taking a look at the above picture 1.3, we can now imagine the three different infrastructures as three different companies. Since there is no shared storage, which all of them needs to be connected to, these are now completely separated which brings a lot more flexibility. We can now see ourselves in a position where we don't know at all what type of hypervisor that runs at one company, and then try to move the VM from the source to the "unknown" destination.

1.2 Problem Statement

The given problem stated in the section above 1.1.2 shows a possible future of how it could be possible to move virtual machines across heterogeneous cloud environments without using a shared storage. Since the task above will not use any type of shared storage and prevent vendor lock-in it will bring a lot of flexibility into how virtual machines can operate at different locations. Which means that the administrator working at one company may have a variety of choices in which he choose what type of hypervisor to perform different tasks. This may have huge advantages as some hypervisors could be more dedicated at doing some tasks than others. It may also give the opportunity to deploy virtualized infrastructures to other companies without having to think about what type of hypervisor that runs at the "other end".

Therefore, the problem statement in this project will be:

How can we move virtual machines across heterogeneous cloud environments without common dependencies, like shared storage, in order to gain flexibility and prevent vendor lock-in?

To solve the problem it will be necessary to at least setup two completely separated and independent clouds. As to show the intention of the problem statement it would also be preferably to have different virtualization architecture layer at these clouds. The clouds will be maintained from a third party host, most probably a local machine, through *SSH*.

The environment will be chosen to be at one location, instead of separating the two clouds between, for example, two colleges. The reason for choosing this is the probability of having network and firewall issues and also to save time setting the environment up and running. As the environment will be highly isolated it will leave me in more control of the two clouds.

As this environment will not have any form of a shared storage, the migration can not be done live as mentioned in section 1.1.2. The discussion of *Cold Migration vs Live migration* is therefor necessary and will be taken into account in section 2.3.

1.3 Thesis outline

This document will be structured as follows:

- **Chapter 1:** Introduces the motivation and goal of the thesis, and takes a look at what the todays problem in the field really is.
- **Chapter 2:** Presents necessary background information relevant to the topics and goals of the thesis.
- **Chapter 3:** Explains the approach, which will discuss the design and the setup process to complete the function of the thesis.
- **Chapter 4:** Explains the detailed software setup on all the involved physical servers.
- **Chapter 5:** Presents the data and the test results. Shows important factors such as implemented scripts, with important omitted core code and functionality, and test diagrams and corresponding graphs.
- **Chapter 6:** Analyzes and discuss the results. Looks at the different VM's behavior in multiple execution scenarios and forms two estimated functions. Comes up with an efficiency theory towards VM down-time.
- **Chapter 7:** Draws a conclusion to finalize the thesis work.

Chapter 2

Background and literature

This chapter will provide a short introduction to some of the topics that is relevant or in correlation with this project. It will also show a general overview on *Virtualization* and *Cloud Computing* and highlight the most common advantages and disadvantages. It will also outline the most common software solutions.

2.1 Virtualization

In the 1960's virtualization was developed to partition large, mainframe hardware to utilize hardwares in a simple and modular way. During the 70's there were still a lot of positive hopes and determination to increase the sharing and utilization of the expensive mainframe resources. Over in the 80's virtualization saw certain fallback due to the decrease in hardware costs. Which caused the organizations to move and replace the centralized mainframes into minicomputers. The determination and motivation for virtualization was therefor not as attractive as before. Into the 90's virtualization came back on track as Daniel A. Menascé [4] describes:

"The advent of microcomputers in the late 80's and their widespread adoption during the 90's along with ubiquitous networking brought the distribution of computing to new grounds. Large number of client machines connected to numerous servers of various types gave rise to new computational paradigms such as client-server and peer-to-peer systems."

In 1974, Gerald J. Popek and Robert P. Goldberg published a paper entitled "Formal Requirements for Virtualizable Third Generation Architectures"[5]. In this paper, they introduced three essential characteristics for a VMM (Virtual Machine Monitor):

"First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources."

Today the term *Virtualization* can have a variety of "definitions" where there exists no formal one which is in general agreed upon. We can although think of virtualiza-

tion as a way to allow several different operating systems to run individually on one physical host. Meaning that they all share the same available hardware resources while running simultaneously on one single computer.

Virtualization has become a base requirement for organizations when moving to a dynamic infrastructure, and because it delivers abilities that is not possible in physical environments. Like, presenting one single physical resource as many individual logical ones or make numerous physical resources to appear and function as one single logical device. The ability to provide either of those phrases, will give benefits like operational automation, resource optimization and a high level of availability, which is not possible in physical servers. Virtualization can also decrease the number of physical servers and therefor reduce the cost of cooling, power and data center space.

2.1.1 Virtualization categories

Virtualization itself is an old technology, but since the hardware and OS have matured to the point where virtualization becomes an advantageous tool in todays society, it gains more and more popularity. There are several ways to implement server virtualization. In accordance to [6, 7] the two leading approaches are:

- Full virtualization
- Paravirtualization

There also exists a third popular approach, which is called *Operating System Virtualization*. Since we are not touching in on this any further, it will not be more thoroughly discussed.

Full virtualization

Full virtualization is designed to provide a total abstraction of the underlying physical system and create a complete virtual system in which the guest operating system can be executed. This approach uses a software called a *hypervisor* to create this abstraction layer between the virtual servers and the physical underlying hardware.

With this abstraction the guest OS (virtual server) is not aware of its virtualized environment. Which means that it allows for any OS to be installed on the virtual server without performing any modifications.

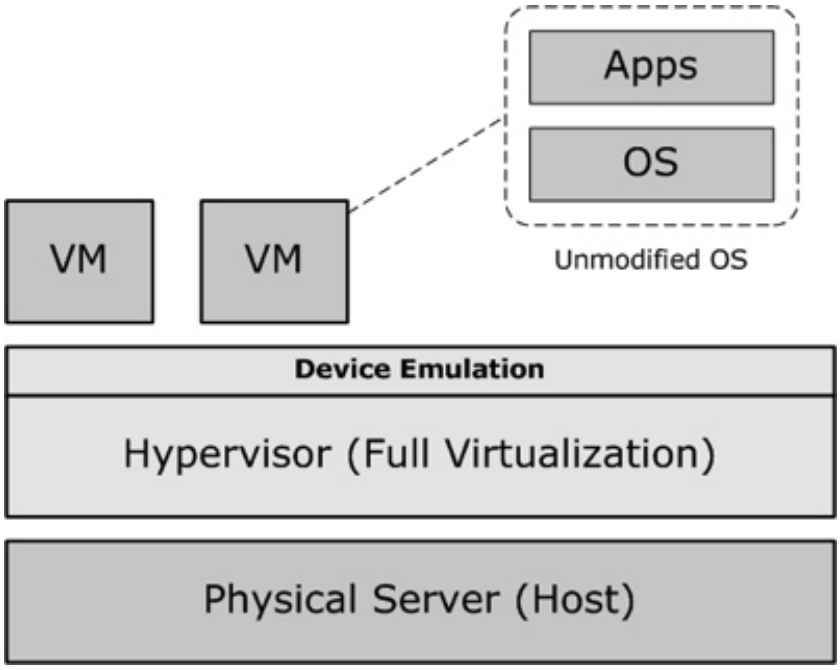

Figure 2.1: Full virtualization Architecture

Main advantages of using *full virtualization* is its easy setup/installation and that it enables complete decoupling of the software from the hardware. This will help to provide a complete isolation of different applications which will make it very secure.

Since the hypervisor itself demands some processing power it will naturally generate some performance penalty from the physical server, as this has to reserve some amount of power to the hypervisor application. As a result this could slow down other running applications. The hypervisor also needs to emulate the virtual servers and act as a bridge towards the physical resources. Which of course makes it quite complex.

Common software examples of this approach is the commercial VMware ESX Server[1] and the open source solution Kernel Based Virtual Machine (KVM)[2].

Paravirtualization

As we can see from the section above, the issue with *full virtualization* is the emulation of devices within the hypervisor. With paravirtualized solutions the guest OS is aware that it's being virtualized. To reduce the burden on the hypervisor from its operations in fully virtualized systems, is to modify each running guest operating systems so that they know they are running in virtualized environments.

[1] http://www.vmware.com/products/vsphere/esxi-and-esx/index.html
[2] http://www.linux-kvm.org/page/Main_Page

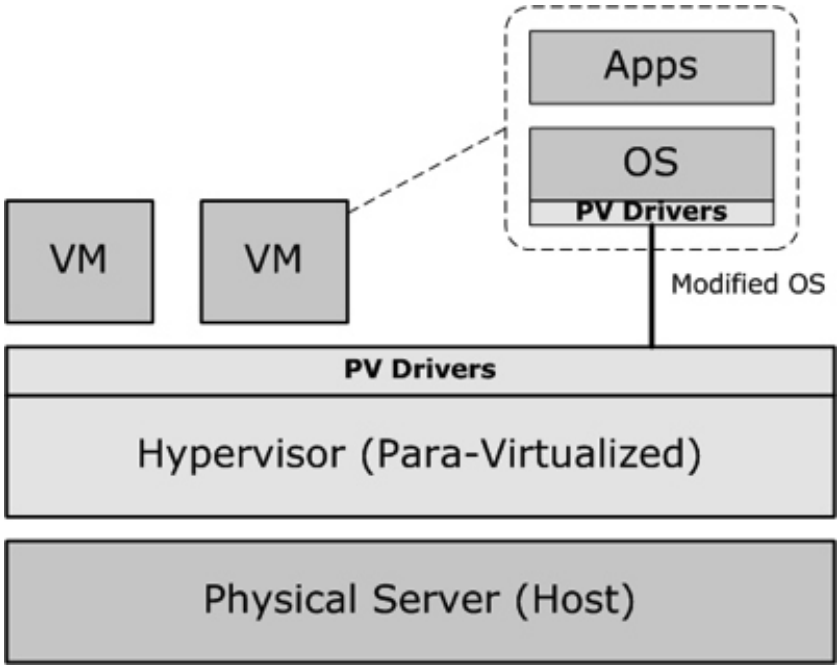

Figure 2.2: Paravirtualization Architecture

In paravirtualization the low-level emulation of the devices is removed, and replaced with cooperating guest and hypervisor drivers. The advantage with this approach is the overall performance since the hypervisor and guest OS is running on cooperated drivers, but the disadvantage is that all the guest OS's must be modified to integrate hypervisor awareness.

The most well known solution for implementing *paravirtualization* is RedHat's Xen[3]. As the following paper implies [8]:

"Xen is one example of an open source para-virtualization technology. Before an OS can run as a virtual server on the Xen hypervisor, it must incorporate specific changes at the kernel level. Because of this, Xen works well for BSD, Linux, Solaris, and other open source operating systems, but is unsuitable for virtualizing proprietary systems, such as Windows, which cannot be modified."

Although today, most virtualization solutions supports paravirtualization in some way. Solutions like; VMware[4], Microsoft Hyper-V[5] and KVM[6].

Conclusion

From the above discussion one can loosely conclude that what you gain with flexibility using *Full virtualization* you loose in performance compared with *Paravirtualization*.

[3]http://www.xen.org/
[4]http://www.vmware.com/products/vsphere/esxi-and-esx/index.html
[5]http://www.microsoft.com/en-us/server-cloud/hyper-v-server/default.aspx
[6]http://www.linux-kvm.org/page/Main_Page

2.1.2 Other types of Virtualization

Virtualization is a widespread technology and an expression that covers different ways to virtualize the variations of resources that are available. The standard server virtualization has been discussed above, but there are a few more which are shortly presented below:

- *Desktop virtualization* or *client virtualization* is a method to separate the desktop environment and store the "virtualized" desktop on a remote server. Users will hereby have the ability to access their applications, processes and data using stateless thin clients [9].
- *Storage virtualization* is a way to consolidate multiple network storage devices into what appears to be a single storage unit [10, 11]. Hereby applications will no longer need to know where to find the data on any specific drives or partitions. This concept also helps the automation of storage capacity expansion (that is: expanding storage resources without any manual provisioning).
- *Network virtualization* makes it possible to combine multiple network resources and administrate them into one single unit, called a virtual network [12]. Having this single collection of resources, allows any authorized user to share network resources from a single computer.

2.1.3 Advantages and Disadvantages of Virtualization

Since the start of implementing virtualization, there have been discussed new ideas and ways to have advantages of using this as an addition to or a replacement of physical resources. The following advantages are the most characteristic and well known today [4, 13, 14]:

- *Availability*: A feature when it comes to server/desktop virtualization, which is not available in physical environments, is *Live Migration*[15]. The ability to live migrate a resource from one to another, when for example performing maintenance, without shutting the server down.
- *Isolation*: A virtual machine is isolated from other virtual machines and hosts. When it comes to attacks, this has a lot of security advantages. One attack does not compromise other virtual machines and therefor leaking of data is also not an issue.
- *Partitioning*: The advantage of reducing the number of physical servers. Hereby splitting the one large resource into smaller, but similar "chunks". Doing this has the benefit of reducing power consumption and air conditioning costs.
- *Infrastructure savings*: There is no need to employ extra infrastructure, when there is a need for extra software or hardware. Virtualization allows users to create virtual environments that suits the specific needs and can therefor use the same infrastructure for different purposes.
- *Responsiveness*: The ability to respond rapidly to computing requirements and the change of needs within an organization.

- *Flexibility*: For system administrators, virtualization gives them more possibilities when designing, configuring and maintaining systems. Together with easy deployment tools, administrators can get rid of redundant manual tasks.
- *Manageability*: Using a wide range of virtualization implementations, administrators can remotely manage and create virtual machines.

As we can see, virtualization has several obvious advantages, but there are also some clear disadvantages as well, and the most important one is the single point of failure. When virtualizing several services on one physical host, all the running services and virtual machines will go down when the physical server gets compromised.

2.2 Cloud Computing

Cloud computing have existed for a lot of years, and a lot longer than most of us think. Once the word Cloud became a modern trend in the 21st century, we thought that "this is a whole new world opening up". This without noticing that some of the parts had already existed for many years, for example *Hotmail*. *Hotmail* was probably not the first application for users to access over the Internet, but it was at some point one of the most widespread and most used. Today *Gmail* is exactly the same running application at the top level of the Cloud stack as *Hotmail* was for about 15-20 years ago.

Cloud computing really came on track as the virtualization technology increased in the 90s. Because it leveraged all the virtualization capabilities, but presented it as a service out to customers over the Internet. In that way it opened up a lot of doors for businesses to run their infrastructure out on some vendors datacenter instead of hosting the whole thing itself. It also raised the possibility for developers to upload and run programs and code at some other platform, instead of doing this locally, which also gave the opportunity to access this program wherever there were an Internet connection.

In 2009, Peter Mell and Tim Grance presented the paper "Effectively and Securely Using the Cloud Computing Paradigm" [16], in which they came with a clear suggestion on how to define cloud computing in general:

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable and reliable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal consumer management effort or service provider interaction."

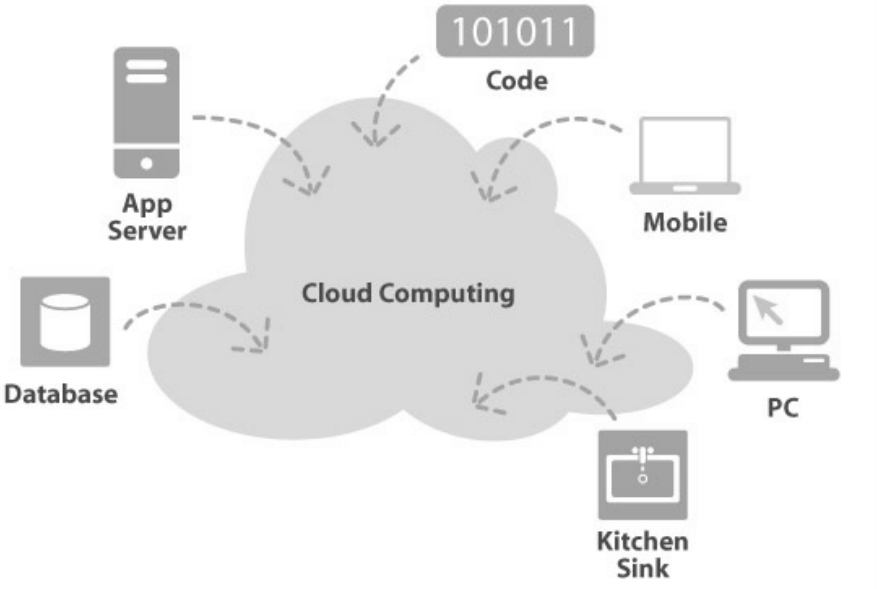

Figure 2.3: Overview of the vast capabilities when using Cloud Computing

2.2.1 Deployment models

There are certain differences in which types of clouds one can run, in terms of business specifications and technical requirements. We categorize them into four cloud deployment models: private cloud, community cloud, public cloud, and hybrid cloud. According to the paper "Cloud Computing: Deployment models, delivery models, risks and research challenges" [17], this is how each of the models are defined:

- *Private cloud*: Enterprise owned or leased. The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.
- *Community cloud*: Shared infrastructure for specific community. The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.
- *Public cloud*: Sold to the public, mega-scale infrastructure. The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.
- *Hybrid cloud*: Composition of two or more clouds. The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

2.2.2 Delivery models

Cloud computing providers offer their services according to three fundamental delivery models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS):

- *Infrastructure as a Service (IaaS)*: This is the base layer of the cloud stack and most probably what we think of when we hear the word "Cloud Computing". The most known vendors are: *Amazon EC2 and S3* and *Sun Microsystems Cloud Services*. At this level ones get to choose one or more virtual machines, and configuring them with your kind of CPU, RAM etc to suite your needs. The customer will pay on an hourly or monthly basis and only for the resources that that has been consumed. The biggest portion of this payment is just uptime, whereas network bandwidth or I/O operations are more or less non-existent price-wise.

- *Platform as a Service (PaaS)*: This layer is the middle layer in the stack, and is mostly used by developers. The most known vendors are: *Google App Engine*, *force.com* and *Microsoft Azure*. We can think of this layer as an environment for developers to run, test and store code, applications and programs. Instead of buying very expensive licenses to maintain a code-platform locally, it is provided as a service in the cloud.

- *Software as a Service (SaaS)*: You guessed it right. This is the top layer in the cloud stack. Aimed to provide applications over the network. Cloud clients will not have the access to the platform or the infrastructure in which this application runs. At this level, we don't have to think the "cloud"-way anymore. Let's make it more visualized and simple. *Facebook* or *Gmail*. "Everyone" uses one or both of these services every day, and who would have thought that these are actually running applications at the top level of "the cloud".

2.2.3 Advantages and Disadvantages of Cloud Computing

As Cloud Computing has become a very popular utility and trend in moderns years, it has been discussed a lot of ways on how this can replace the physical running infrastructure. The following advantages and disadvantages are the most characteristic and well known today [16, 17] :

- *Multi-tenancy*: In a cloud environment, services owned by multiple providers are co-located in a single data center. The performance and management issues of these services are shared among service providers and the infrastructure provider. The layered architecture of cloud computing provides a natural division of responsibilities: the owner of each layer only needs to focus on the specific objectives associated with this layer.

- *Cost effective*[7]: Cloud computing is often cheaper and less labor-intensive for companies too. There is no need to buy and install expensive software because it's already installed online remotely and you run it from there, not to mention

[7] http://www.shapingcloud.com/the-cloud/what-are-the-benefits-of-cloud/

the fact that many cloud computing applications are offered free of charge. The need to pay for extensive disk space is also removed. With cloud computing, you subscribe to the software, rather than buying it outright. This means that you only need to pay for it when you need it, and it also offers flexibility, in that it can be quickly and easily scaled up and down according to demand. This can be particularly advantageous when there are temporary peaks in demand, such as at Christmas or in summer, for example.

- *Unlimited storage*[7]*:* A major advantage of using cloud computing for many companies is that because it's online, it offers virtually unlimited storage compared to server and hard drive limits. Needing more storage space does not cause issues with server upgrades and equipment - usually all you need to do is increase your monthly fee slightly for more data storage.

- *Easy access*: Clouds are generally accessible through the Internet and use the Internet as a service delivery network. Hence any device with Internet connectivity, be it a mobile phone, a PDA or a laptop, is able to access cloud services. Additionally, to achieve high network performance and localization, many of todayâĂŹs clouds consist of data centers located at many locations around the globe.

- *Dynamic resource provisioning*: One of the key features of cloud computing is that computing resources can be obtained and released on the fly.

Running the infrastructure in public clouds have, as described above, a lot of advantages, but there are a few disadvantages as well:

- *Downtime/Loss of control*: If your rented infrastructure goes down in some way or a loss of internet connection, you as a system administrator will have no way, but to wait, to bring this back up again, which can be really critical and time and productivity consuming.

- *Data security*: Storing data on third-party servers will always lead to privacy and confidentiality issues. This because you will have no control or knowledge on how the data is secured at these servers.

- *Latency*: Often you have no idea where those data centers that holds your data are placed. If the data center is located far away the client connection time may not be as fast as you thought it might be.

2.2.4 Public cloud providers

Figure 2.4: A collection of some of the major public cloud providers

The figure above 2.4 shows that there are a lot of different public cloud providers out there today. Some of them provide PaaS clouds and other provides IaaS clouds (these where discussed in 2.2.2).

The most well known PaaS cloud providers are Google App Engine[8] and Windows Azure[9]. While the most popular IaaS clouds are Amazon EC2[10], GoGrid[11] and Rackspace[12]. As this thesis has its main focus on *Infrastructure as a Service*, these clouds will be discussed later on as a potential to move towards hybrid cloud solutions.

[8] https://developers.google.com/appengine/
[9] http://www.windowsazure.com/en-us/
[10] http://aws.amazon.com/ec2/
[11] http://www.gogrid.com/
[12] http://www.rackspace.com/

2.2.5 Private cloud providers

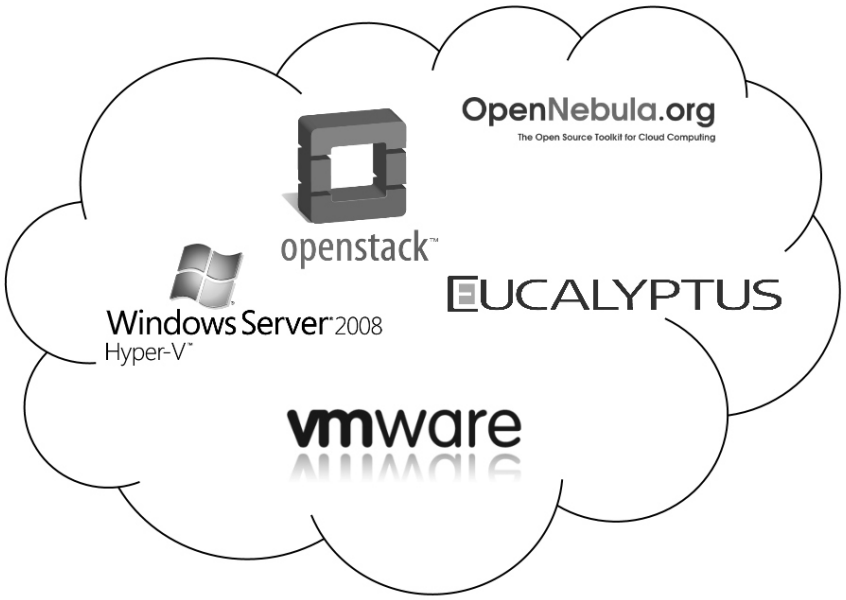

Figure 2.5: A collection of some of the major private cloud providers

Today there exists numerous private cloud providers. All with the aim to let companies install and setup these solutions easily by themselves. Some of the providers lets the users choose what virtualization layer to run upon, e.g. Xen or KVM. Other providers are more lock-in, which often makes it more simple to set up and easy to know what distinct tasks where they have their strengths.

Providers like VMware[13] and Microsoft Hyper-V[14] has their own proprietary solutions and both are well known and widespread. While OpenNebula[15], OpenStack[16] and Eucalyptus[17] are all open source and easily changeable when it comes to the underlying running virtualization technology (hypervisor).

In this thesis the main focus will be upon OpenNebula. More on the configuration and setup will follow in the "Methodology and theory" and "System Setup" chapters.

[13] http://www.vmware.com/solutions/cloud-computing/index.html
[14] http://www.microsoft.com/en-us/server-cloud/private-cloud/default.aspx
[15] http://opennebula.org/
[16] http://openstack.org/
[17] http://www.eucalyptus.com/eucalyptus-cloud

2.2.6 OpenNebula

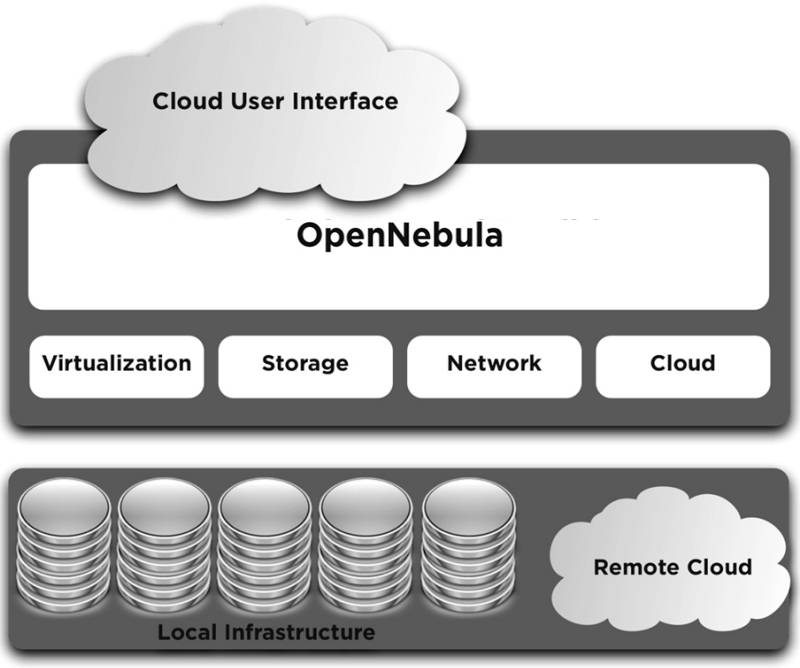

Figure 2.6: The OpenNebula Architecture

For managing heterogeneous distributed datacenter infrastructures, OpenNebula is probably the most well established open source tool available. OpenNebula first came on track in 2008 and their latest release 3.4 was released in April 2012. In this project version 3.2 will be used.

There exists three objectives which makes OpenNebula well suited for this project:

- Openness of the architecture, interfaces, and code
- Interoperability and portability to prevent vendor lock-in
- Standardization by leveraging and implementing standards

The most important point here, is the *Interoperability and portability*. By this OpenNebula makes it possible to install different private clouds using different hypervisors. OpenNebula has integrated support for KVM, Xen and VMware. As was mentioned in the problem statement, this feature will test the "no vendor lock-in" aspect as two clouds running OpenNebula will run Xen and KVM as hypervisors.

Today there exists a lot of alternatives when choosing which private cloud to host an infrastructure. Many of these were presented in the above section 2.2.5. Along with OpenStack, OpenNebula is one of the major private IaaS cloud providers which is based on *Free and Open Source Software (FOSS)*.

At the start of this project OpenStack was installed and configured, to solve this thesis problem statement. OpenStack is a very upcoming and new open-source project, which led to a genuine interest to explore further into this thesis. Although after some time, it was clear that OpenStack had some bugs and errors which often led to a restart of multiple services or as worse as a total re-installment of the whole private cloud itself.

After tedious hours spent on configuring and maintaining the OpenStack cloud, I was introduced to OpenNebula. The basic architectural principals of both clouds were the same. A nice and simple GUI Front-End and back-end compute nodes with a corresponding CLI.

The reason why OpenNebula was chosen at the end, in favor of OpenStack, was its quality and matureness. The installation process is perhaps somewhat more detailed and nitpicking, but the result is more robust and there were far less errors encountered throughout the rest of the project.

2.3 Cold Migration vs. Live migration

The basic concept of migration is to move for example a virtual machine from one host to another. When Live migration was introduced one of the main differences was the "down-time". The less down-time, down to the unnoticeable, was the approach of achieving Live Migration.

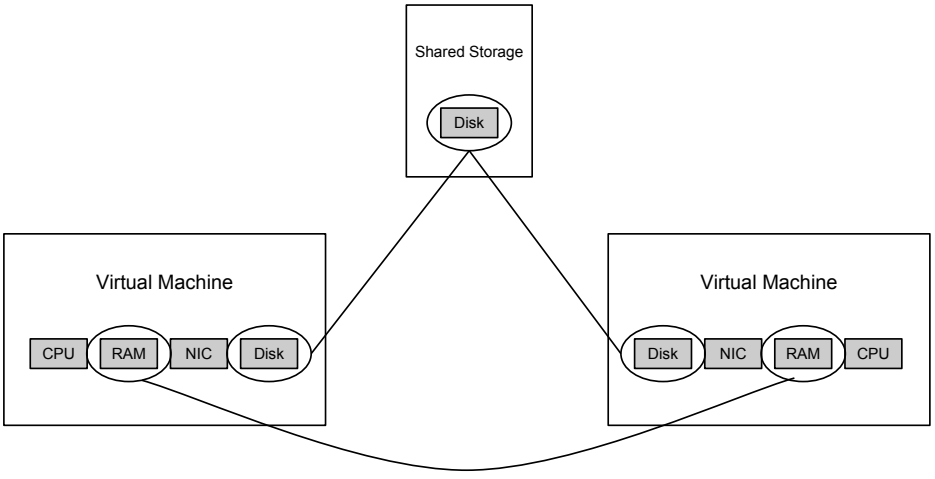

Figure 2.7: Basic Principals of the Live Migration Process

One other important thing was the shared storage dependency (as discussed in section 1.1.2). Having a shared storage made it possible to migrate the state of the machine, instead of the disk image which stays at the storage and is available from the

destination host as well. The state of the virtual machine is the memory (RAM). So when one wants to live migrate a virtual machine, it is the memory of the machine that is being moved from one host to another.

2.3.1 Memory Migration Steps

When live migrating memory of a virtual machine, there are certain stages in the process:

- The first thing that happens is that the hypervisor copies the current memory from source to the destination, while the virtual machine remains running at the source.
- Once the memory is copied, the virtual machine can be stopped at the source, which initiates what we know as the "down-time".
- After the source VM is stopped, memory differences from when the memory copy started until the source VM was stopped, is copied to the destination. These "extra" memory-pages is called dirty-pages.
- Once the VM at the destination is started with all memory-pages copied, the migration is completed and the "down-time" is finished.

During a *live migration*-process the "down-time" is often just a few milliseconds, but it depends of course on the size of the dirty-pages and running applications. When users does not feel any glitch in the services it is often called a *seamless live migration*.

2.3.2 Advantages and Disadvantages

When performing a "Cold Migration"[18], for example in cases where you do not have a shared storage, you have to move the actual disk image. In many cases these are quite large and can take quite some time to copy from source to destination. Which in return may result in a potentially large down-time from when the virtual machine is stopped at the source, until it is booted up at the destination.

Another downside is that the memory/RAM will not be copied, meaning that running applications will not be remembered at the new location.

By choosing to migrate the disk image instead of the state, it opens up the opportunity to move virtual machines across heterogeneous environments. All cloud implementations that support a virtual machine boot based on a raw disk format will be able to replicate virtual machines from other environments. And this ability exists today in all standard and popular cloud providers like OpenNebula, OpenStack, VMware and Hyper-V.

Chapter 3

Methodology and theory

In this chapter the approach will be explained. It will cover the basic design of the experimental environment, including:

- Hardware equipments and software tools
- Infrastructure design
- Planned scenario

3.1 Objectives

The problem statement was previously discussed in 1.2 at page 5. It will now be reiterated here in a more formal manner, based on terms and concepts introduced in the background chapter.

How can we migrate virtual machines across heterogeneous private clouds without common dependencies, eg. storage, in order to gain flexibility and prevent vendor lock-in?

This project will first of all setup and maintain two independent clouds (both OpenNebula), with a different architecture (Xen and KVM). Both clouds will be in a controlled environment and managed by an independent localhost.

As it will not involve any shared storage, the migration will take form in what we could call a "cold migration".

Having the understanding of both cloud environments it will be possible to migrate virtual machines from one cloud to the other, with the aim of doing this as efficiently as possible. Efficiently will mean to come up with solutions to find a good way that will migrate the machines from one location to the other.

It could also be preferably to look at *policy decision making*, as this will bring the solution more towards a realistic scenario.

3.2 Environment

The project will be implemented and managed at *Oslo and Akershus University College of Applied Sciences*. All equipment to perform the project are available at this location. This will have the benefit of not having to use time to travel between different locations, and not having issues like network downtime and connectivity and also trouble with multiple firewalls outside the College network. The environment will be isolated and the actual design will be thoroughly discussed in section 3.3.

3.2.1 Physical Servers

The following table 3.1 shows the basic technical details of the configured servers in this project:

Hostname	CPU	Memory
Controller1	Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz	2GB
Node-01	AMD Phenom(tm) 9550 X4 CPU @ 2.2GHz	32GB
Node-02	Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz	2GB
Controller2	Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz	2GB
Compute-01	AMD Phenom(tm) 9550 X4 CPU @ 2.2GHz	32GB
Compute-02	Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz	2GB

Table 3.1: Technical server information

More on the roles of the servers will follow in section 3.3 and 3.3.1.

3.3 Infrastructure Design

The infrastructure design will consist of two clouds located at *Oslo and Akershus University College of Applied Sciences*. Both clouds will be running OpenNebula and consist of 1 controller and 2 compute nodes. More on the roles of each server will be shown in 3.3.1.

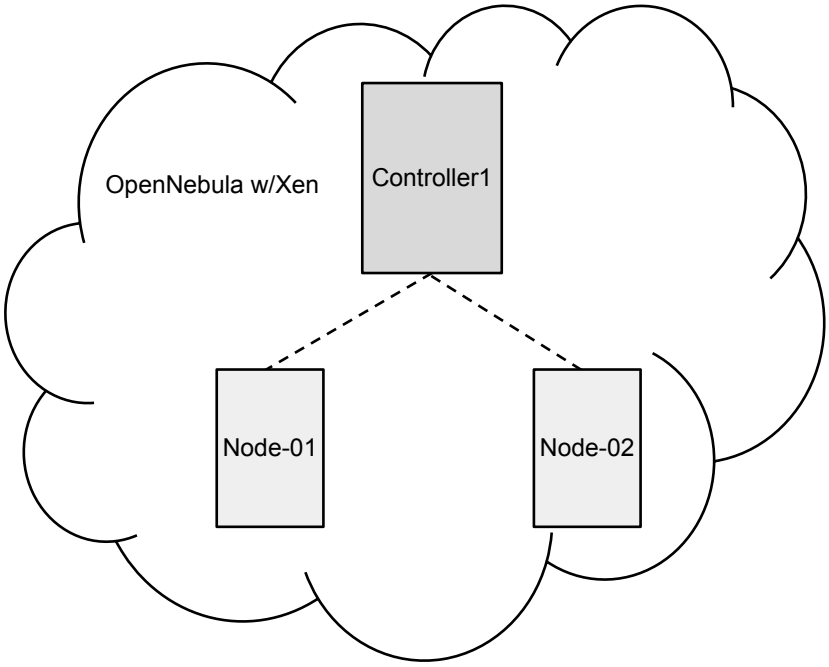

Figure 3.1: Cloud1 w/Xen

There will be one cloud (alias: Cloud1), that will run OpenNebula with the Xen hypervisor. It will have two compute nodes behind the controller which will take care of the virtual machines. The actual installation and setup of these clouds will be presented in the "System Setup" chapter at page 4.

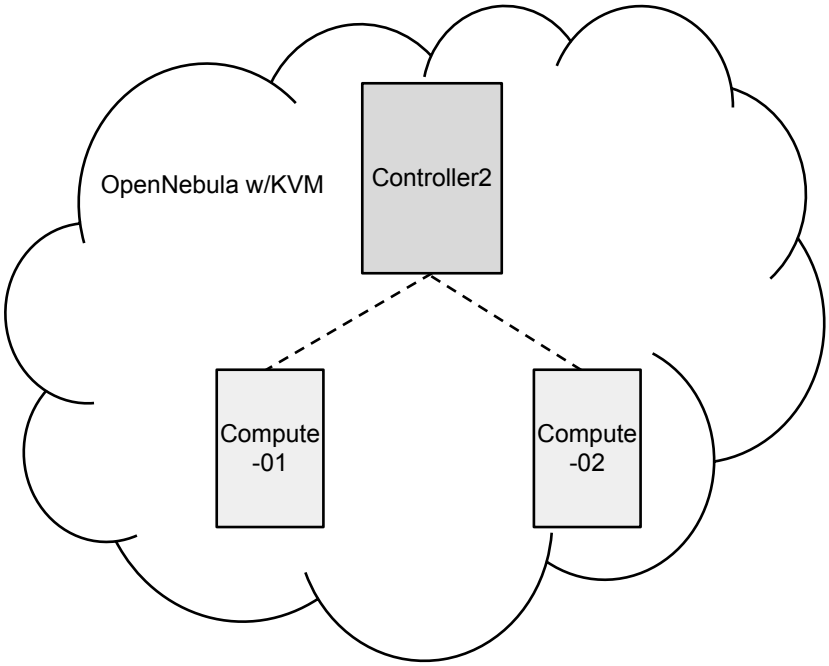

Figure 3.2: Cloud2 w/KVM

The second cloud (alias: Cloud2) will be similar as Cloud1, except for the hypervisor, which in this case will be KVM.

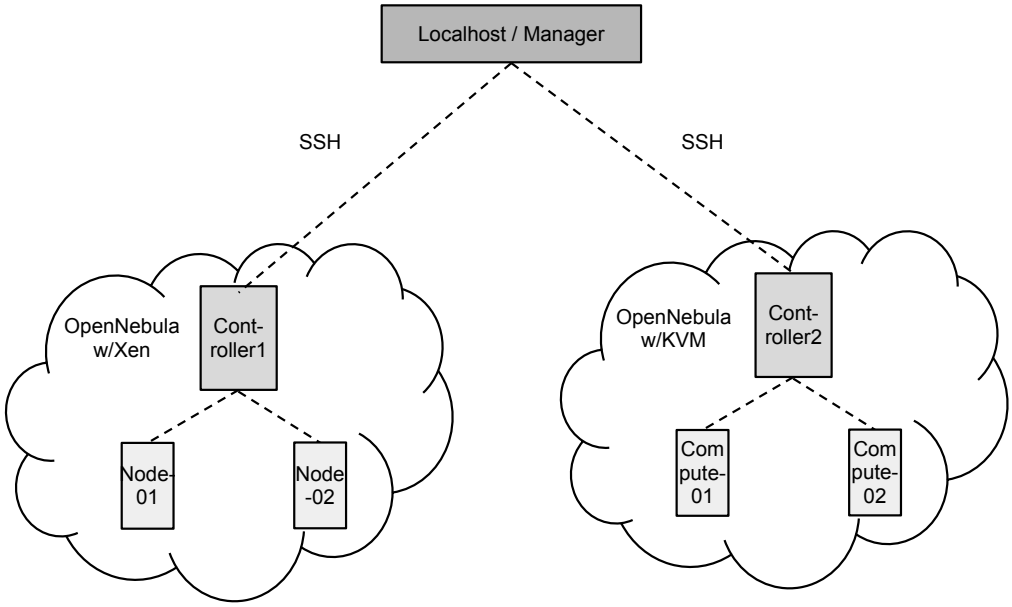

Figure 3.3: Overview of the Infrastructure Design

Both clouds will be administrated and coordinated from a third machine. The machine will have *SSH*-access towards both of the clouds, which means that it will have command line access and can manage them locally, albeit separately.

3.3.1 Server and Technical Environment

The table below 3.2 shows the basic info of the running servers in the test environment. Their IP address and role are the important factors:

Hostname	**IP address**	**Software**	**Description**
Controller1	128.39.74.2	OpenNebula Front-End	Cloud1 controller
Node-01	128.39.74.10	Xen	Compute node on Cloud1
Node-02	128.39.74.11	Xen	Compute node on Cloud1
Controller2	128.39.74.29	OpenNebula Front-End	Cloud2 controller
Compute-01	128.39.74.20	KVM	Compute node on Cloud2
Compute-02	128.39.74.21	KVM	Compute node on Cloud2

Table 3.2: Server roles and software

The two clouds will be separated by one 100Mbit switch. The reason for not choosing a gigabyte switch, actually on purpose, is because the network at *Oslo and Akershus University College of Applied Sciences* is so fast, so the switch holds it back so that the performed tests will take some amount of time. This is why a basic Ubuntu image on 2GB is chosen as well, instead of a ttylinux with 45MB.

3.4 Scenario

As discussed earlier in the "Introduction" chapter at page 3, without a shared storage ones have to migrate the disk image instead of the state/memory of the VM. Therefor the focus of this project will be to successfully move the disk image from one cloud to the other in an efficient manner. It will also be useful to make a replicate of the CPU and RAM.

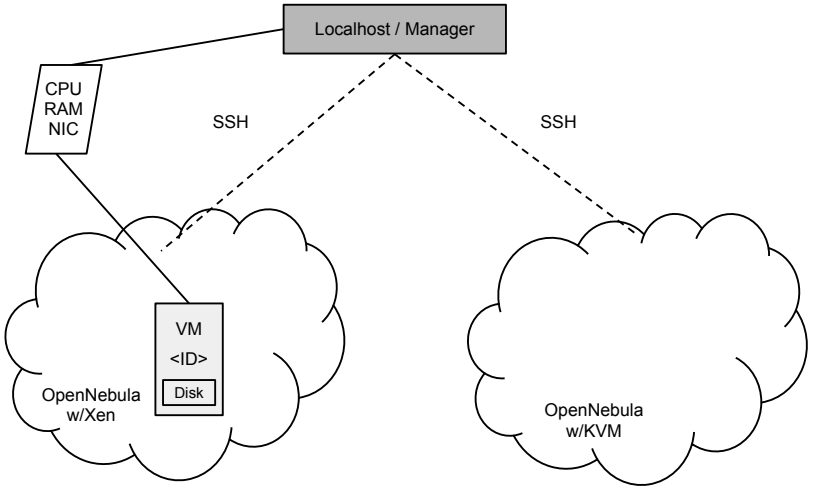

Figure 3.4: Execution Scenario Step 1

Lets take a possible migration from Cloud1 to Cloud2 as an example. First of all it will be necessary for the localhost (alias: Manager) to get the basic info of the running VM from Cloud1. The most important info will be the amount of RAM and CPU, so that it will be possible to make a replicate of this information on a new VM at Cloud2. It will also be necessary to get some information on the NIC as well.

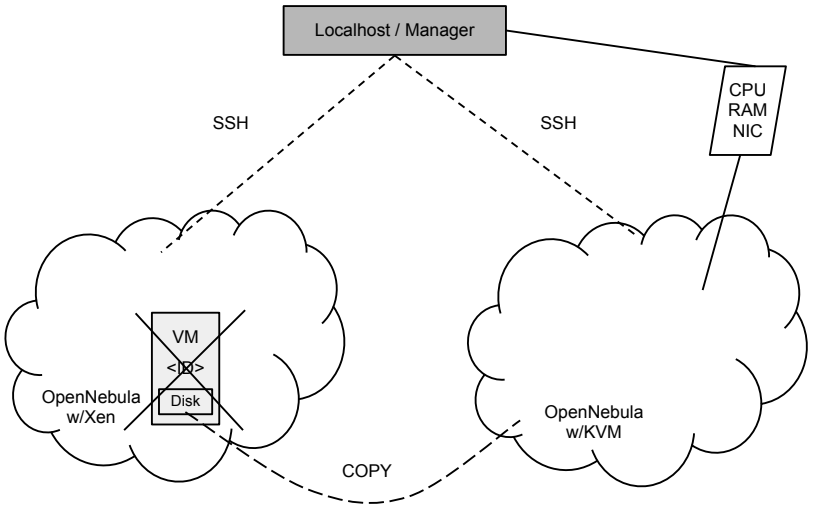

Figure 3.5: Execution Scenario Step 2

Once this info is stored and sent as a template file towards Cloud2, the VM at Cloud1 can be stopped, and the copy of the disk image can be initiated. Meaning that the user can not perform any more operations toward its VM and the down time for the user has therefor started.

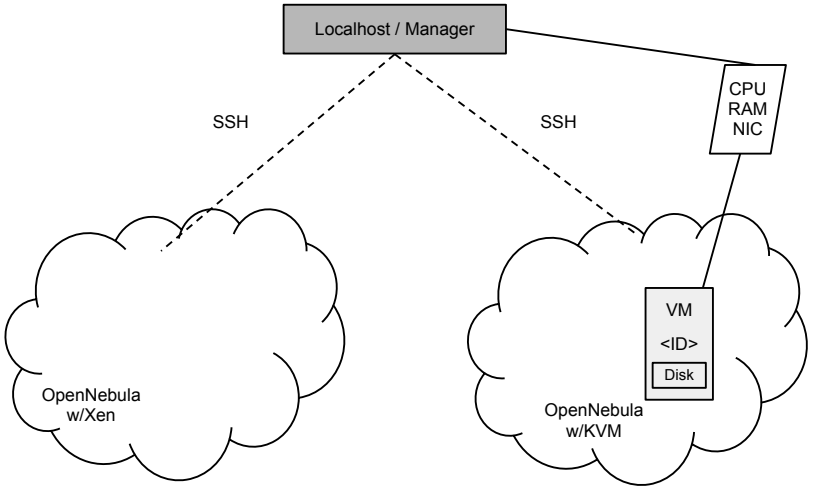

Figure 3.6: Execution Scenario Step 3

When the copy of the disk image is succeeded over to Cloud2, a new VM can be started based upon the copied disk image and the template file with the replicated information from the previous VM at Cloud1.

As mentioned in section 2.3, the downside of this type of migration will be that the state of the machine will not continue, and all running applications has to be re-initiated on the new cloud.

3.4.1 Development

To be able to perform such a task as presented above, it will be obvious to create some form of script which will gather the template info, and initiate both the copy stage and the VM execution stage. As Perl has simple modules to establish SSH-connections, and is well known to be fast and efficient when handling file operations such as read and write, it will probably be a good choice of tool to use.

The following example could be a decent way to perform the execution of the script, although the parameters may vary as the script develops. The script will be further presented and discussed in section 5.2 at page 34.

Script execution example

```
./script -i [VM-ID] -p [password]
```

The two parameters above will indicate which VM to migrate, identified by its ID number. The script will also have to handle the password so that it would be possible to get access.

As presented in 3.1 it could be interesting to do this in an automated way, so that we

can look at more policy based decision on when to do the migration decision. Therefor it will be necessary to implemented an automation script which will be executed based on time or date, and for example specify the decision on distinct users.

The following is how an example of how this execution may look like:

Automation script execution example

```
./autoscript -u [user]
```

Also here the parameters may change as the script develops, and in section 5.3 at page 38 this script will also be further presented.

Chapter 4

System Setup

In this chapter the system implementation is presented, including the installation and configuration of OpenNebula and the compute nodes.

All machines that are involved in the environment at *Oslo and Akershus University College of Applied Sciences* are running Ubuntu 11.10. The focus of this project is not on the networking side. So to be sure, and since this project is only a prototype, all machines and VM's are applied static IP's from a given subnet.

4.1 Installing and setting up OpenNebula

Different versions and the newest stable release of OpenNebula can be found here [19]. OpenNebula itself has some Ruby libraries requirements, which they have made a script to detect common linux distributions and install the required libraries. The location for the script is */usr/share/one/install_gems*. In this project case, the script did not find the packages needed, the following packages had to be installed manually:

- sqlite3 development library
- mysql client development library
- curl development library
- libxml2 and libxslt development libraries
- ruby development library
- gcc and g++
- make

After installing all dependencies, there are some basic configuration in terms of user modification and secure shell access required [20]. When all this is properly configured, the control node is completely installed.

I will not go into detail on how to setup the front-end web interface as it is not that relevant for this project to work properly, but it is very useful to have when working

on and maintaining the cloud infrastructure. Basic installation and configuration of the Sunstone web interface can be found here [21].

4.1.1 Compute nodes

At the compute nodes there are no requirements to install any OpenNebula components. These are the only requirements at the host machines:

- ssh server running
- hypervisor working properly configured
- ruby 1.8.7 or newer

4.2 Using the KVM hypervisor in OpenNebula

If KVM is the hypervisor that will run, which it is on one of the clouds in this project, this is how the /etc/one/oned.conf should look like (this is the default in OpenNebula):

File: /etc/one/oned.conf

```
IM_MAD = [
    name       = "im_kvm",
    executable = "one_im_ssh",
    arguments  = "-r 0 -t 15 kvm" ]

VM_MAD = [
    name       = "vmm_kvm",
    executable = "one_vmm_exec",
    arguments  = "-t 15 -r 0 kvm",
    default    = "vmm_exec/vmm_exec_kvm.conf",
    type       = "kvm" ]
```

Standard KVM is full-virtualized, so to make KVM paravirtualized along with Xen, it is necessary to use the VirtIO[1] framework. As presented in section 2.1.1, paravirtualization outperforms full virtualization performance wise, therefor it is vital that both clouds in the upcoming test scenarios are equal. The VirtIO framework supports a para-virtual Ethernet card and a para-virtual disk I/O controller, which makes KVM comparable to Xen.

4.3 Using the Xen hypervisor in OpenNebula

As described earlier in 3.3, one of the clouds will run the KVM hypervisor, which is standard in OpenNebula, while the other will run Xen. Therefor OpenNebula needs to know if it is going to use the Xen hypervisor. To achieve this, uncomment these drivers in /etc/one/oned.conf:

[1] http://www.linux-kvm.org/page/Virtio

Small changes in /etc/one/oned.conf

```
IM_MAD = [
    name       = "im_xen",
    executable = "one_im_ssh",
    arguments  = "xen" ]

VM_MAD = [
    name       = "vmm_xen",
    executable = "one_vmm_exec",
    arguments  = "xen",
    default    = "vmm_exec/vmm_exec_xen.conf",
    type       = "xen" ]
```

All compute nodes must have a working installation of Xen that includes a Xen aware kernel running in Dom0 and the Xen utilities. Detailed information on the installation and configuration of Xen can be found at [22].

There is no need on the compute nodes to install any OpenNebula components. This also applies when running the Xen hypervisor, of course.

4.4 Configuring local shared storage

Both clouds will have local shared storage. To make the infrastructure simple, the controller on each cloud will function as the shared storage for the compute nodes. NFS will be used so that the nodes and the controller with the shared storage is connected. They can therefor access files and disk images across the network as if they resided in a local file directory. Basic installation and configuration of the NFS-server can be found at its web site [23].

At the controller, we need to specify the distinct folder which to be shared among the backend compute nodes. In this case the shared storage is mounted in */var/lib/one*, and all directories under this will be shared as well.

4.5 Common OpenNebula CLI commands

To get a short overview on the OpenNebula CLI commands, the most common ones are presented below:

Submits a new VM, adding it to the ONE VM pool.

```
onevm create <template>
```

Shuts down VM by its ID.

```
onevm shutdown <vm_id>
```

Stops a running VM.

```
onevm stop <vm_id>
```

Suspends a running VM.

```
onevm suspend <vm_id>
```

Deletes a VM from the ONE VM pool.

```
onevm delete <vm_id>
```

Gets information about a specific VM.

```
onevm show <vm_id>
```

Lists all VM's.

```
onevm list
```

Chapter 5

Results

This chapter covers the experiment output and the final results. The following information is presented:

- Developed scripts and log-files
- Environment test graphs
- Parallel vs Sequential migration

The following sections present the data primarily in graphical form. Some omitted versions of log-files and scripts will also be presented. See the Appendixes for full versions of the developed scripts.

5.1 Developed Scripts and Their Functions

There exists several scripts that has been developed to realize the migration scenario and get proper results from the executions. The name of the scrips and their functions, with corresponding log files are listed in the table below:

Function	Script name	Log file name
One VM migration	migration.pl	migration_log.txt
Automation	automation.pl	auto_log.tsv
Logging destination cloud	dest-log.pl	dest_log.tsv
Parallel migration	parallel.pl	log_par2.txt log_par3.txt log_par4.txt
Parser	log-parser.pl	log_par2_parsed.txt log_par3_parsed.txt log_par4_parsed.txt log_seq2_parsed.txt log_seq3_parsed.txt log_seq4_parsed.txt
Generating byte load	disk-load.pl	*Unkown*

Table 5.1: Developed Scripts and Their Functions

5.2 Developing the migration tool

As was mentioned earlier, in 3.4.1 at page 27, a tool had to be developed to gather and join all the necessary migration stages in a coherent manner. The following options were implemented:

All available options

```
./manager.pl -h

Usage:
-s      Source IP
-D      Destination IP
-p      Source Password
-P      Destination Password
-u      Source User Name
-U      Destination User Name
-i      Virtual Machine ID from Source Cloud
-o      Path to Logfile
-h      Help
-v      Verbose
-d      Debug
```

The above flags are all vital and necessary to be able to execute the script, except for *-o, -h, -v* and *-d*. The script performs and does as described in 3.4, as the following core code will obviously explain:

manager.pl: Important core code

```
1 @info_template = get_vminfo_from_cloud1();
2 my @mod_template = modify_template(@info_template);
3 create_template_on_cloud2();
4
5 shutdown_vm_on_cloud1();
6 copy_disk_from_cloud1_to_cloud2();
7
8 create_vm_on_cloud2();
9 delete_vm_on_cloud1();
```

The general thought and flow of the script happens in these methods. The example code above is from a scenario where ones wants to migrate from Cloud1 to Cloud2. First of all the script gets the information of the running VM, get_vminfo_from_cloud1(). Then it modifies the current running VMs template, so that it fits the running environment in Cloud2, modify_template(). It will then create the new modified template on Cloud2, create_template_on_cloud2(). Once the template is copied to the destination, the running VM on the source will be shut down, shutdown_vm_on_cloud1(). After the shut down is completed, the copy of the VM's disk to the destination can be initiated, copy_disk_from_cloud1_to_cloud2(). The destination Cloud will after the disk copy have access to both the previous running disk image on Cloud1 and also the template file which specified the RAM, CPU and NIC etc. This means that the script can now create a new VM on the destination based on the previous disk and template, create_vm_on_cloud2(). Once the new VM has been created, the script can delete the old VM at the source, to save disk space on the source. delete_vm_on_cloud1().

The execution of the script will look like the following:

manager.pl execution example

```
./manager.pl -v -d -s 128.39.74.2 -D 128.39.74.29 -p Dolly -P Dolly /
             -u admin -U admin -i 100 -o migration_log.txt
```

The example execution above takes in the IP of the source cloud and the IP of the destination clouds. It also specifies the passwords for each of them (ex. Dolly) and uses admin as the user on both clouds. The VM ID from the source cloud, in which will be migrated to the destination cloud, is set to 100. Also a logfile, which is optional, is specified at the end (migration_log.txt).

5.2.1 Test results

As the manager.pl script runs it logs the time as the script starts and also when it is finished, which means right before the template info storage and after the new virtual machine is running. These numbers are then saved in the log file for every execution. The following example is an omitted version of the output file (migration_log.txt) from the above execution:

migration_log.txt: omitted output example from manager.pl

```
1332706523      1332706638
1332706754      1332706870
1332707018      1332707133
1332707232      1332707346
1332707768      1332707882
```

To be able to get out some proper data from the numbers above, a simple parser was made, which can be found in the Appendixes (log-parser.pl). The same parser will be used for different logfiles, which is why it was made, as it will effectively produce more precise numbers when creating graphs and diagrams. The following example shows how the new output file looks like:

out.txt: omitted output example from parsing migration_log.txt

```
115
116
115
114
114
```

The numbers above indicates, in seconds, how long the end-to-end time will be for the user when migrating from one cloud to the other.

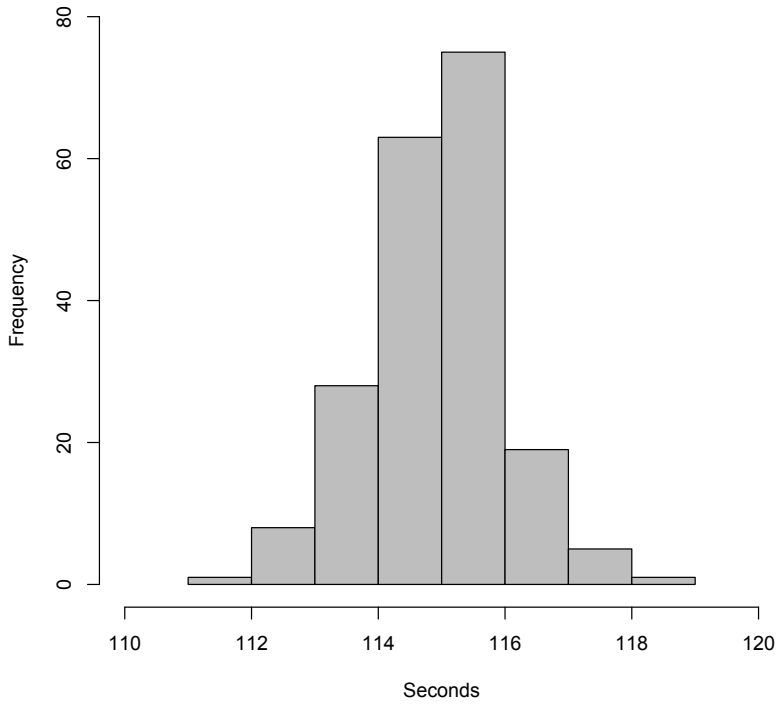

Figure 5.1: 200 migration executions of one single VM

The graph above is the result from traversing the above explained log file, after executing one single migration from source to destination 200 times. It indicates a behavior in which the approximate average is around 115 seconds or 1:50 minutes. This also corresponds quite correct when looking at the corresponding omitted log file above.

The important side of this is that the above time, is the end-to-end time, the total time from when the three different stages in the migration process starts until they are finished:

- Read/Write Template Stage
    1. get_vminfo_from_cloud1()
    2. modify_template()
    3. create_template_on_cloud2()
- Copy Disk Stage
    1. shutdown_vm_on_cloud1()
    2. copy_disk_from_cloud1_to_cloud2()
- Boot Stage
    1. create_vm_on_cloud2()
    2. delete_vm_on_cloud1()

The *Read/Write Template Stage* includes some simple read and write operations towards the VM corresponding template files. Logging this operation shows that this stage takes around 2 seconds. When the *Copy Disk Stage* starts, the down-time begins. This is the time that may vary, due to network latency, and be the major factor towards the end-to-end time. The *Boot Stage* will also be consistent around 20 seconds, in the following scenarios. The *Read/Write Template Stage* and *Boot Stage* numbers are very easy to find out by just logging each distinct stage itself, and therefor not important to present in any further formats. So when we look at the above scenario, the *Copy Disk Stage* takes approximately 90-95 seconds.

Having the ability to approximately know how long one single VM's migration end-to-end time is from the source to destination is vital for further migration scenarios, which will be further presented later on in this chapter.

5.3 Developing the automation tool

As the "Objectives" section 3.1 mentioned, it would be a good idea to automate the migration process. Which will give the administrators less manual job when executing migrations of multiple VMs. It will also give the administrators a chance to make more policy based decisions on when to perform such a task, by putting for example the automated script in a cron job. The following options was implemented in the automation script:

All available options

```
./automation.pl -h

Usage:
-c source:destination   Cloud IP addresses
-u source:destination   Cloud user names
-p source:destination   Cloud passwords
-t                      Delay time, in seconds, between each VM migration
-o                      Path to Logfile
-h                      Help
-v                      Verbose
-d                      Debug

./script -c ip:ip -u user:user -p pw:pw -t seconds [-o] [-d] [-v] [-h]
```

The options above are quite similar as the options in the previous manager.pl script. Although here the IPs, users and passwords are set using the same flag, but yet separated. The main difference is the *-t* option, which makes it possible to decide how long the time between each migration should be. This could come in handy in many situations, if there are many VMs that needs to be migrated, but as it will use a lot of the network bandwidth, it gives a certain period where the network is open for other processes. It may also be a nice feature if the administrator wants to configure the VMs at the destination as they are completed, but again have enough time to configure them properly before a new VM comes in.

The execution of the script will look like the following:

automation.pl execution example

```
./automation.pl -c 128.39.74.29:128.39.74.2 -u admin:admin -p Dolly:Dolly -t 10 -o auto_log.tsv
```

As the execution example above states, the source and destination IP, user and password are separated using a colon. The delay between each migration is set to 10 seconds and also an optional logfile (auto_log.tsv) is used.

5.3.1 Test results

The basic function of the script is to work in a loop, as many times as the number of VM's that the user, specified in the execution, is responsible for at the source Cloud. For every VM that migrates, the local time and the number of VMs is saved in a log file. The following example is an omitted version of the output file (auto_log.tsv) from

the above execution:

auto_log.tsv: omitted output example from automation.pl

```
21:52:06    5
21:54:11    4
21:56:18    3
21:58:25    2
22:00:31    1
22:02:34    0
```

The above example shows the local time and the number of VMs in the source cloud is logged. What is important to notice about the above numbers is that the local time is the start time for each migration process. Running in parallel with the above automation.pl script is a log script towards the destination cloud. The script is found in the Appendixes (dest-log.pl). The following output example is from the corresponding incoming VMs at the destination:

dest_log.tsv: omitted output example from dest-log.pl

```
21:51:55    0
21:54:00    1
21:56:04    2
21:58:11    3
22:00:16    4
22:02:20    5
```

As the numbers indicates above, the virtual machines is entering the destination cloud pretty much at the same time as the numbers are decreasing at the source cloud.

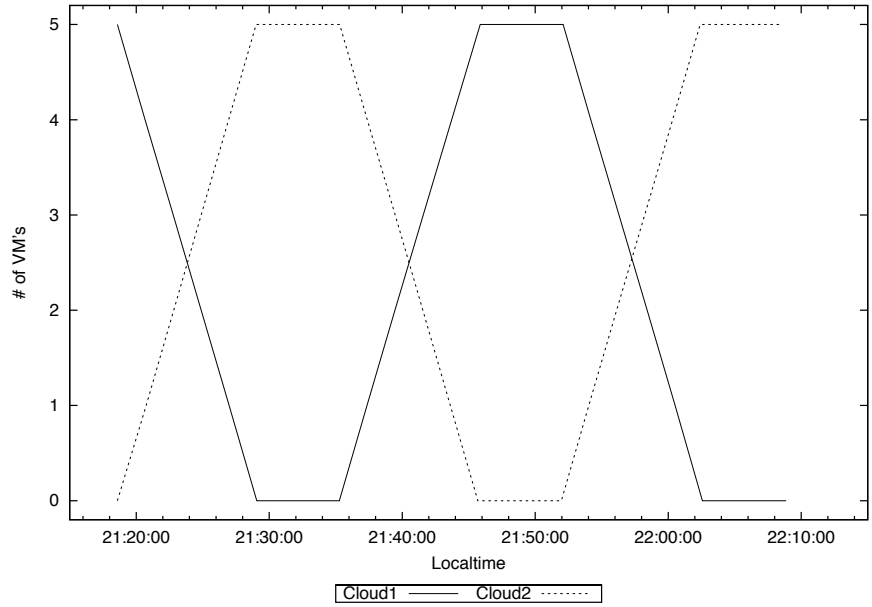

Figure 5.2: Migration between Cloud1 and Cloud2

The graph above shows the flow as virtual machines are being migrated from *Cloud1* to *Cloud2* and vice versa. If we now take a look at the first graph, 5.1 at page 36, which introduced a certain average with 115 seconds for one single virtual machine migration. Then also a short reminder that the delay option was used in this scenario, and set to 10 seconds. Meaning that in this scenario above, each end-to-end time should be approximately *115sec + 10sec = 125sec* or *2:05min*. Which seems to be quite right, specially if we now take a look at the previously introduced and corresponding log files:

Short comparison between auto_log.tsv and dest_log.tsv

```
Cloud1:
21:54:11    4

Cloud2:
21:54:00    1
```

The above example here is just a comparison of the two log files. Here it obvious that the next migration starts 11 seconds after the first virtual machine has been started at cloud2. Which is quite precise using the delay flag and taking some overhead into consideration.

5.4 Multiple Migration Scenarios

There are multiple ways of creating different scenarios when performing a migration from one cloud to another, but there are two basic scenarios that stands out. The first one is parallel migration, which migrates all virtual machines together over the network. The other one is sequential, which migrates one single VM alone and waits until the migration is completed before starting a new migration process.

Testing these to scenarios is vital on determining what's best in either small or large scale deployments. Both scenarios will be tested using 2, 3 and 4 virtual machines with 50 executions each.

5.4.1 Parallel Migration

In parallel migration all virtual machines that belongs to a distinct user are migrated simultaneously, using the same network channel. A short script was implemented to perform such a task, and this is found in the Appendixes (parallel.pl). The following peace of code is the important part of this script to be able to migrate simultaneously:

parallel.pl: Important core code

```
open(CMD1,"./manager.pl -v -s 128.39.74.2 -D 128.39.74.29 -p Dolly -P Dolly /
                                -u admin -U admin -i 100 -o log_par4.txt & |");

open(CMD2,"./manager.pl -v -s 128.39.74.2 -D 128.39.74.29 -p Dolly -P Dolly /
                                -u admin -U admin -i 101 -o log_par4.txt & |");

open(CMD3,"./manager.pl -v -s 128.39.74.2 -D 128.39.74.29 -p Dolly -P Dolly /
                                -u admin -U admin -i 102 -o log_par4.txt & |");

open(CMD4,"./manager.pl -v -s 128.39.74.2 -D 128.39.74.29 -p Dolly -P Dolly /
                                -u admin -U admin -i 103 -o log_par4.txt & |");
```

The above code simply runs the script 4 times, as an example, in parallel as a background process, meaning that 4 virtual machines with ID's 100, 101, 102 and 103 are migrated. The following example is how the log file looks like after the completed execution:

log_par4.txt: omitted output example from manager.pl

```
1333391544      1333392186
1333391544      1333392180
1333391544      1333392184
1333391544      1333392187
```

Once again the log-parser.pl script was used to create more readable numbers, in seconds:

log_par4_parsed.txt: omitted output example from parsing log_par4.txt

```
642
636
640
643
```

An average was calculated for every execution, so that each execution, wether or not it was chosen 2,3 or 4 VM's, ended up having an average end-to-end time. In the above case, with 4 virtual machines running simultaneously, the average end-to-end time was *636.75 seconds*.

Test results

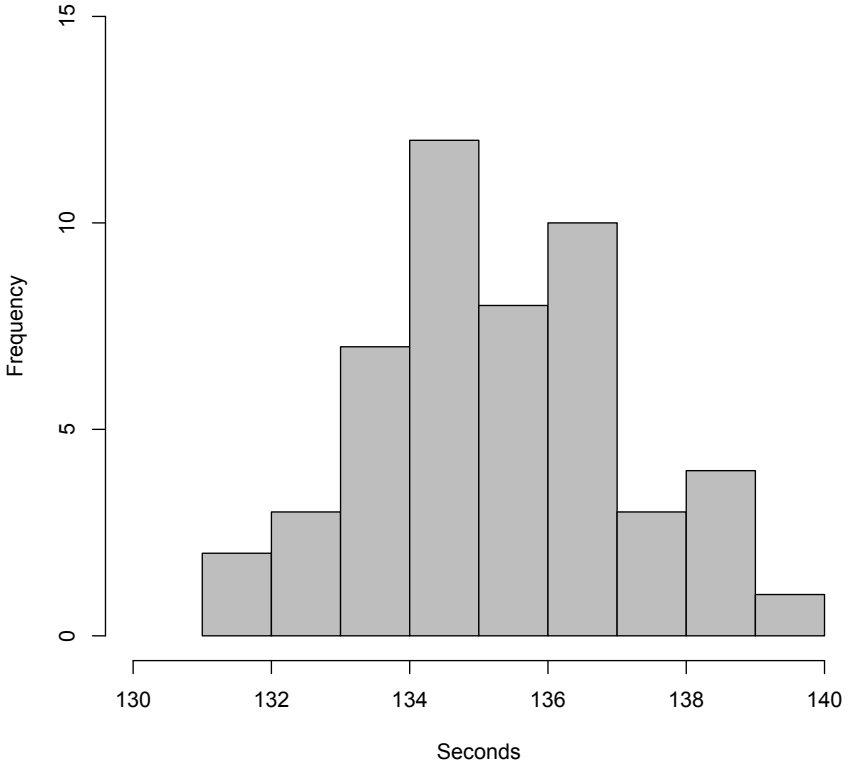

Figure 5.3: Parallel Migration Scenario with 2 VM's

The above diagram shows the results after running 50 executions with 2 virtual machines in parallel. Most numbers are located in the area around 135 seconds or 2:15 minutes. Looking at the first diagram on page 36 where it was a clear indication that one virtual machine takes 115 seconds, two virtual machines should take the same amount of time since they are in parallel, when the overhead is not taken into account. So already when increasing to two virtual machines there exists some overhead (20 seconds) in the network channel. Also worth noticing is the two outliers in the diagram which makes this parallel migration scenario, in very few occasions, unpredictable.

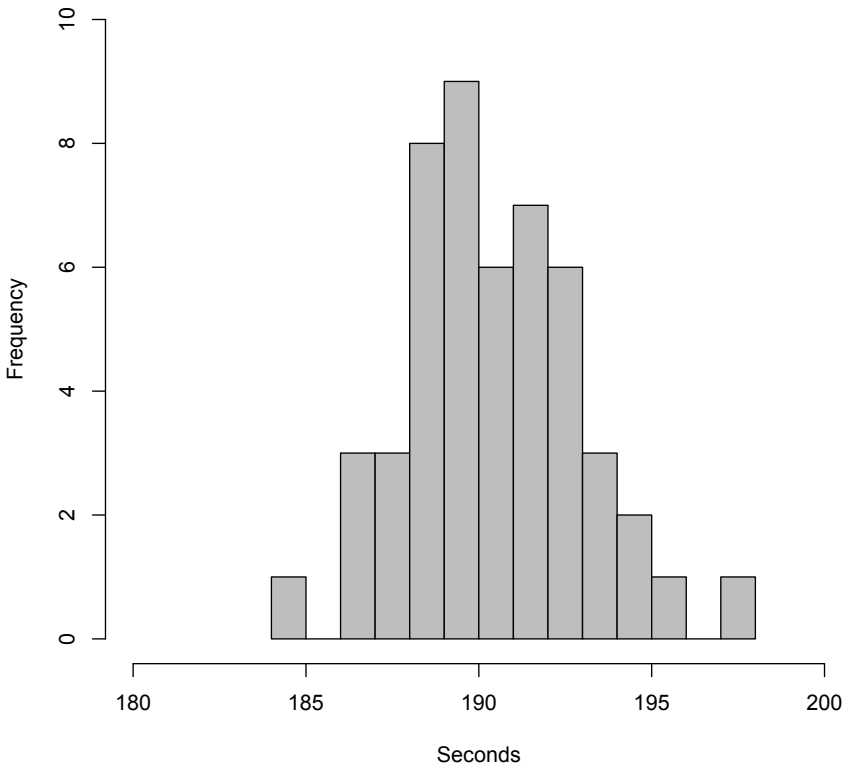

Figure 5.4: Parallel Migration Scenario with 3 VM's

This next graph does the same as the previous only increasing with one VM in parallel. The trend here is that most of the results are located around 190 seconds or approximately 3:10 minutes. This means that the overhead has increased even further, now on approximately 50 seconds, which is quite a countable increase. The variation from minimum to maximum in this diagram is approximately 12 seconds with two small outliers.

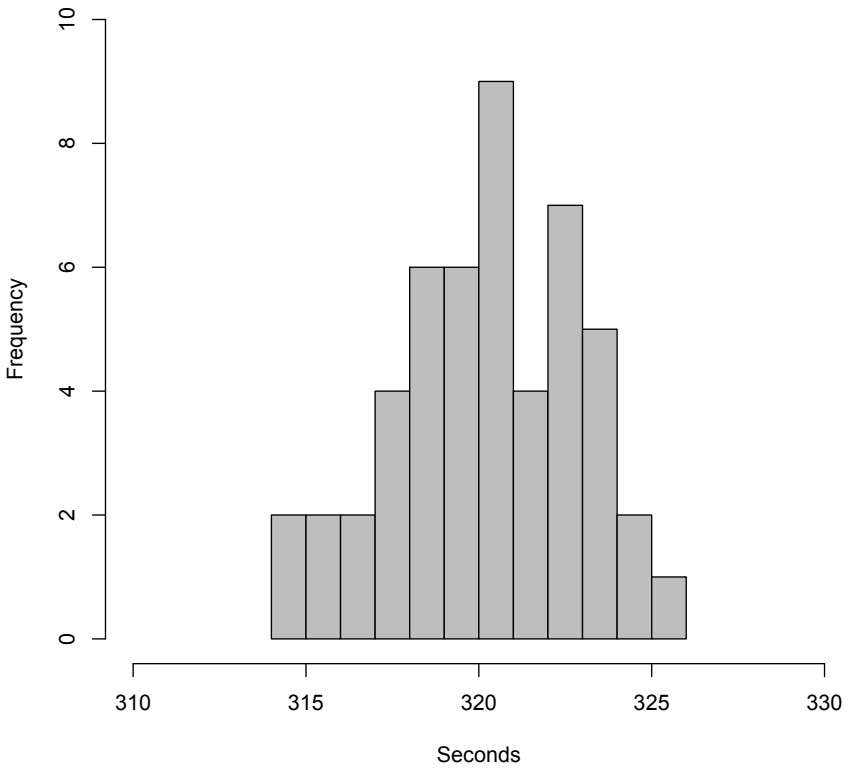

Figure 5.5: Parallel Migration Scenario with 4 VM's

In this last scenario, using 4 virtual machines, the results are located around 320 seconds or 5:20 minutes. With this increase the overhead has increased to approximately 3:25 minutes. This diagram also has an approximate variance of 12 seconds, but here the results are quite compacted.

5.4.2 Sequential Migration

In sequential migration all virtual machines that belongs to a distinct user are migrated in a queue. Meaning that once the first virtual machine has been migrated, the next one can start. There was no need for any script to do such a job, although the following one-liner was used:

Shell running command

```
./manager.pl -v -s 128.39.74.29 -D 128.39.74.2 -p Dolly -P Dolly /
                                -u admin -U admin -i 200 -o log_seq4.txt &&
./manager.pl -v -s 128.39.74.29 -D 128.39.74.2 -p Dolly -P Dolly /
                                -u admin -U admin -i 201 -o log_seq4.txt &&
./manager.pl -v -s 128.39.74.29 -D 128.39.74.2 -p Dolly -P Dolly /
                                -u admin -U admin -i 202 -o log_seq4.txt &&
./manager.pl -v -s 128.39.74.29 -D 128.39.74.2 -p Dolly -P Dolly /
                                -u admin -U admin -i 103 -o log_seq4.txt
```

The one-liner above first migrates VM number 200 and waits until the script is completed, then the next script starts a new process, migrating VM number 201 etc. The following example is how the log file looks like after the completed execution:

log_seq4.txt: omitted output example from manager.pl

```
1333654513      1333654628
1333654643      1333654757
1333654772      1333654887
1333654901      1333655016
```

Once again the log-parser.pl script was used to create more readable numbers, in seconds:

log_seq4_parsed.txt: omitted output example from parsing log_seq4.txt

```
115
114
115
115
```

The results above indicates what was noticed at the graph on page 36, that one VM takes about 115 seconds. Although what is important to notice from the above log file is that there is a delay between each execution, even though they are executed right after the other. This delay between the sequential executions is operations like establishing ssh connections towards both of the clouds. From the next example, the timestamp above has been converted to local times:

Unix timestamp to human date

```
19:35:13      19:37:08
19:37:23      19:39:17
19:39:32      19:41:27
19:41:41      19:43:36
```

The result now show that between each execution it takes around 15 seconds before the next virtual machines end-to-end time starts. This is important when we now are going to look more on the actual results from the different sequential migration tests.

Test results

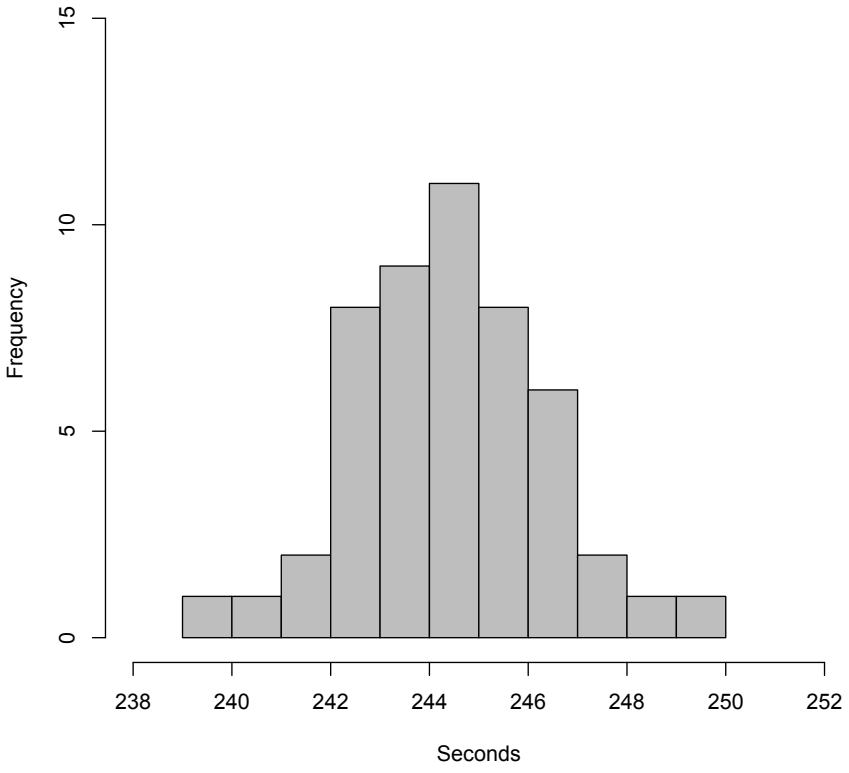

Figure 5.6: Sequential Migration Scenario with 2 VM's

The above diagram shows the results after running 50 executions with 2 virtual machines sequentially. Most numbers are located in the area around 245 seconds or 4:05 minutes. Looking at the first diagram on page 36 where it was a clear indication that one virtual machine takes 115 seconds, two virtual machines should take 230 seconds, when the overhead is not taken into account. This means that using 2 virtual machines, the overhead is 15 seconds, which is quite obvious, because the delay between each sequential migration is 15 seconds. In the diagram above the variation from minimum to maximum is approximately 10-11 seconds.

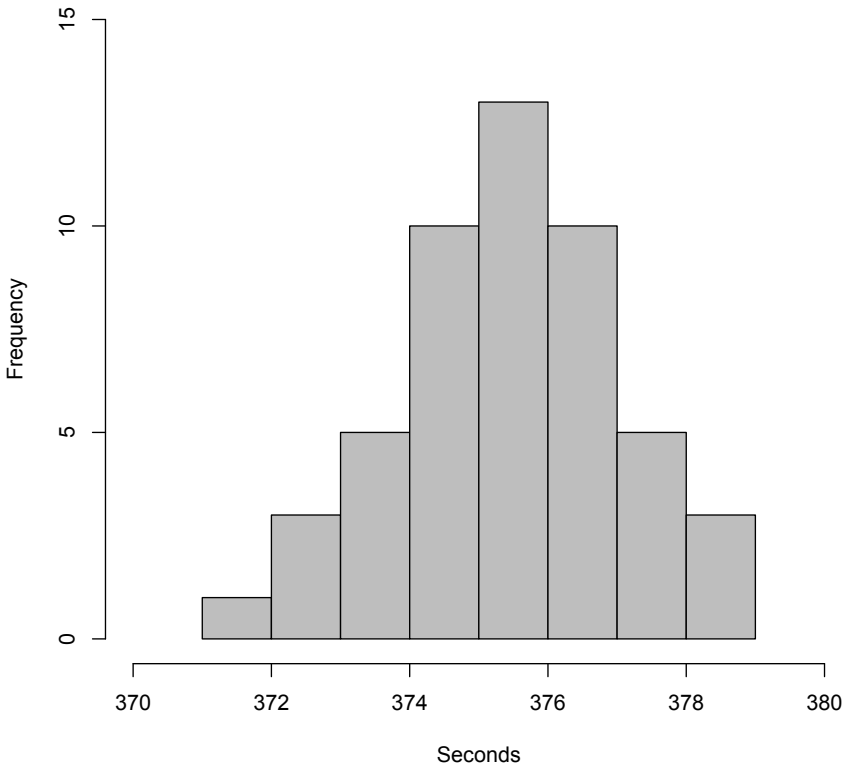

Figure 5.7: Sequential Migration Scenario with 3 VM's

This next graph does the same as the previous only increasing with one VM in "the queue". The trend here is that most of the results are located around 375 seconds or approximately 6:15 minutes. Again looking at the first graph on page 36, migrating 3 virtual machines should result in an average end-to-end time around 345 seconds. Comparing this with the graph above, there exists 30 seconds in overhead, which again is the approximate inter arrival time of 15 seconds between each sequential migration. The results looks slightly more predictable compared to the parallel migrations, following a Gaussian distribution quite closely, with a variation of 8 seconds from minimum to maximum.

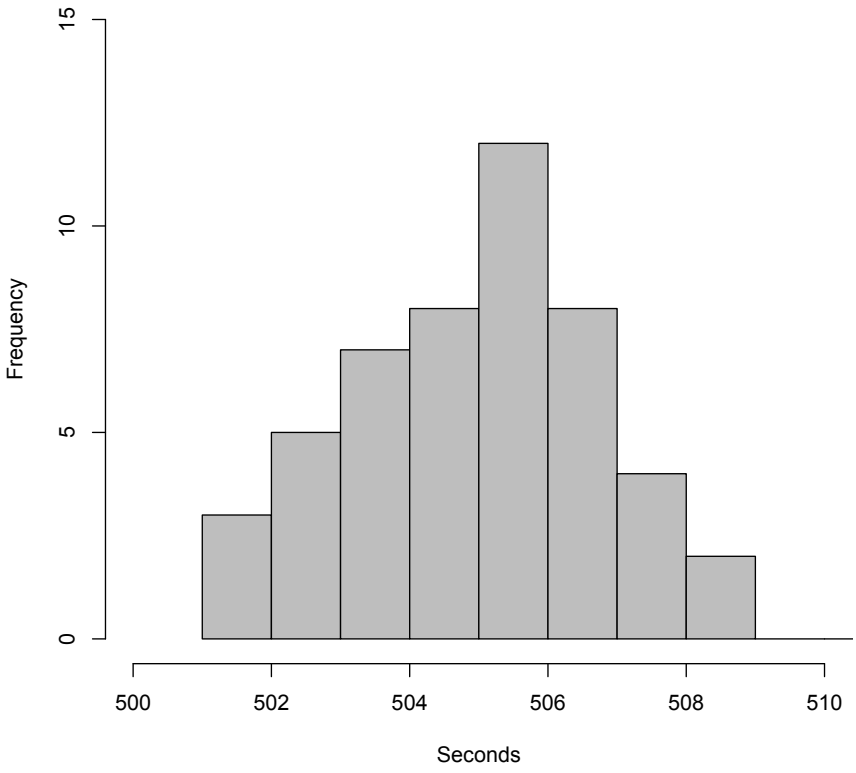

Figure 5.8: Sequential Migration Scenario with 4 VM's

In this last scenario, using 4 virtual machines, the results are located around 505 seconds or 8:25 minutes. With this increase the overhead, compared to running one single virtual machine, which should have resulted in 460 seconds or 7:40 minutes, have increased to approximately 45 seconds. Which obviously, again, is the same as 3x15 seconds inter arrival times.

If we take a look at the automation graph 5.2, at page 40, the virtual machines were also migrated sequentially, yet also with an extra delay time of 10 seconds between each migration process. The following graph is the same as the previous one (5.2), but with a precise marker telling when 4 virtual machines were done migrating.

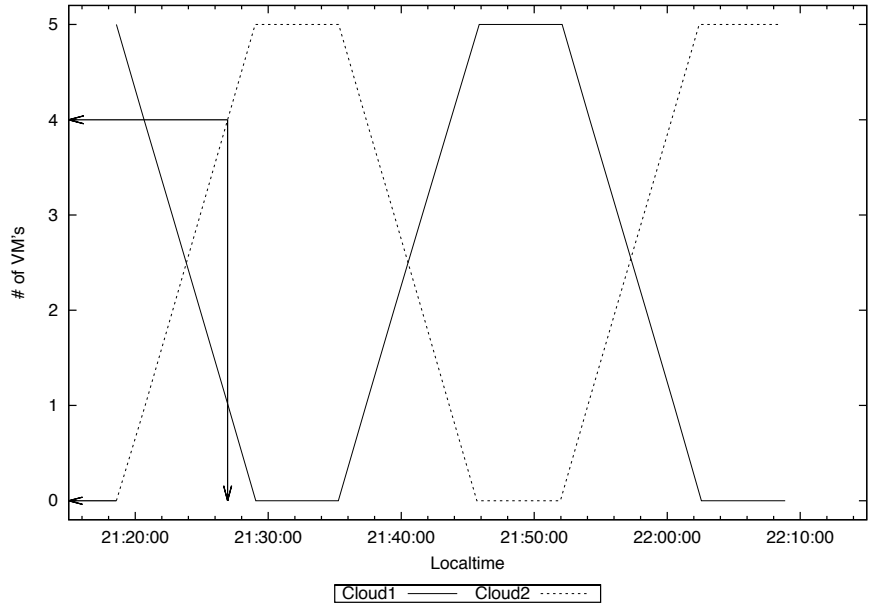

Figure 5.9: Analyzing time consumption when migrating 4 virtual machines

The graph above shows the automation script with an extra 10 seconds delay took right around 9 minutes to complete. If we now compare that to those results above in 5.8, which stated that 4 virtual machines sequentially should take around 8:25 minutes to complete, it seems reasonable when adding the extra 30 seconds inter arrival delay, making it ending up at around 9 minutes.

Chapter 6

Analysis and Discussion

This chapter will cover the analysis and the discussion of the experimental results. It will also introduce another test experiment to form two estimated functions on the different migration scenarios. At last the chapter will introduce some theories in which will make the migration process more effective towards the down-time for the users.

6.1 Virtual Machine Behavior Analysis

As presented throughout the result chapter earlier, the diagrams made a clear indication that the sequential migration scenario had some sort of linear overhead behavior and that the parallel scenario was beneficial time wise, but that the overhead was continuously increasing. Looking more visualized into both scenarios, they are equal in every way in terms of the template stage and boot stage. The main difference lies within the streams of virtual machines towards the network channel between the two clouds.

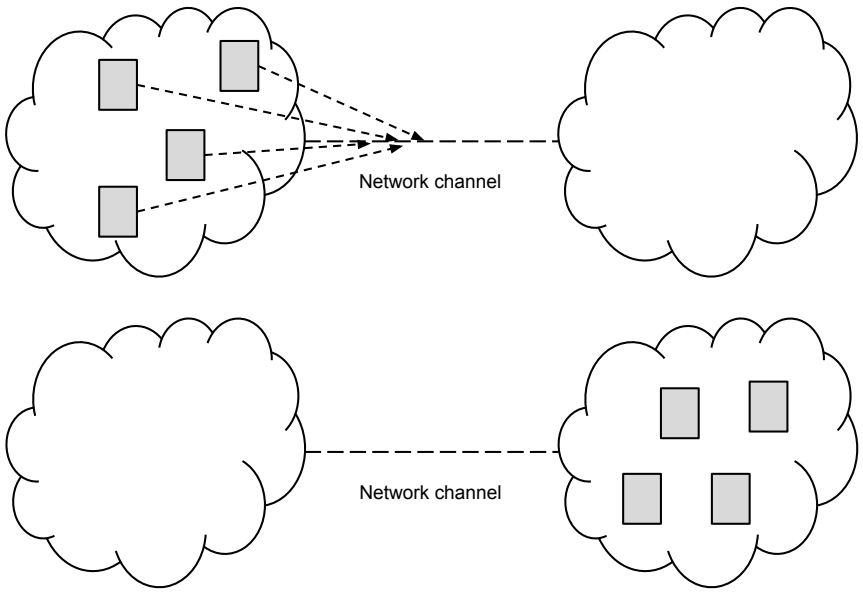

Figure 6.1: Virtual Machine flow in Parallel Migration Scenario

In the parallel scenario, which the Figure 6.1 above shows, all virtual machines is sent on to the network channel at the same time, which will saturate the network as more and more virtual machines are migrated.

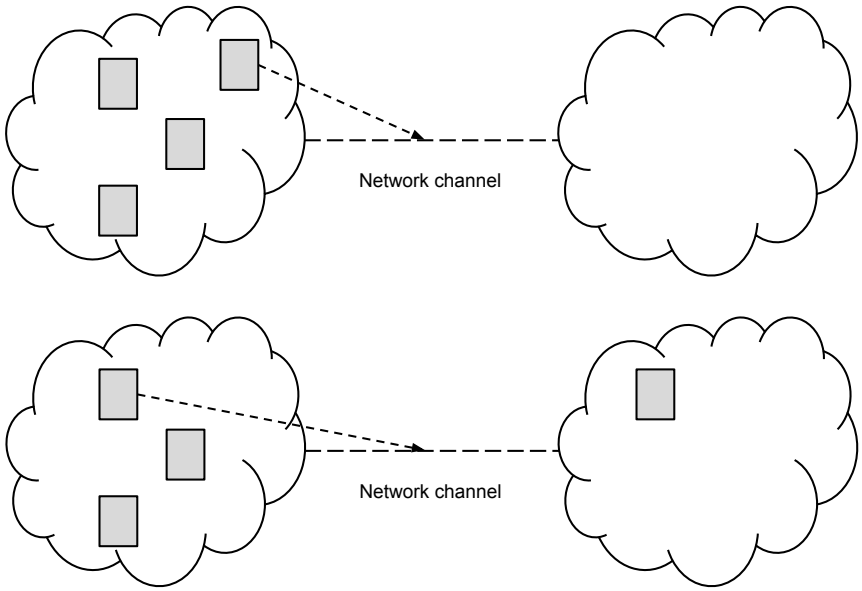

Figure 6.2: Virtual Machine flow in Sequential Migration Scenario

In the sequential scenario, Figure 6.2, every single VM will use the full bandwidth of the network.

Taking a general overview from the test diagrams in the Results chapter, the results clearly indicates that the overhead in the parallel scenario is continuously increasing. It is somewhat clear that the overhead has an indication towards an exponential growth. Compared to the sequential scenario, the only overhead is the delay time between each execution, which tends to be linear. In the next two sections, 6.1.1 and 6.1.2, the above scenarios will be tested and analyzed further, to see if the potential theory stated here is a fact. The sections will also try to come up with a general function for each scenario which will give an approximate indication on the total end-to-end time no matter how many VM's one needs to migrate.

6.1.1 Estimated Function on Sequential Migration Behavior

If we firstly take a look at the sequential scenario, which is the easiest one. The following table will give an overview on the calculated mean for each dataset, from the results, with corresponding variance and standard deviation:

Dataset	**Mean** μ	**Variance** σ^2	**Standard Deviation** σ
2 Sequential	243.90	3.88	1.97
3 Sequential	374.92	2.64	1.62
4 Sequential	504.58	4.04	2.01

Table 6.1: Statistical Overview of the Sequential Migration Datasets

Each dataset contains 50 distinct tests. The reason why it is necessary to perform such an amount, is to make the numbers representative. The *Central Limit Theorem*[1] states that the distribution of a collection of numbers tends to follow a normal distribution when the collection holds 30 or more numbers. Which means that if one wants to gain sufficient statistical results from a collection of numbers, this collection size needs to be 30 or more. Having a collection of 50 tests makes the results even more precise, which makes it easier to conclude the virtual machines behavior.

From the graph above it is now possible to state that for the first test, 95% of the results were located in the area of *243.90sec +- 1.97sec*. For the second test the results are *374.92sec +- 1.62sec* and *504.58sec +- 2.01sec* for the last one.

Given the earlier results, it looks like the sequential migration follows a linear distribution. From this an estimated function can be stated and further tested. The following functions is an idea to how the sequential migration distribution looks like:

$$f(x) = 115x|_{x=1}$$

$$f(x) = 115x + 15(x-1)|_{x>1}$$

Having the theory functions above, further tests can be based on these functions and se if the tests corresponds to the above equations. Since the previous tests have been in quite small scale, the next one will migrate 16 VM's from source to destination. From the above equation, x is how many VM's that will be migrated, with the outcome to be in how many seconds the whole migration process may take.

$$x = 16$$

$$f(16) = 115 * 16 + 15(16-1)$$

$$f(16) = 2065$$

From the equation, migrating 16 VM's should take approximately 2065 seconds or in local time 34:25. The reason for saying "approximately" is because 115 seconds is a

[1] http://en.wikipedia.org/wiki/Central_limit_theorem

general average when migrating one single virtual machine. This time may vary with around 4 seconds when looking at diagram 5.1 at page 36. Due to time consumption, there is not enough time to perform as many as 50 tests using 16 virtual machines. So the following tests will have 30 distinct executions, with 16 VM's migrated in sequentially.

Test results

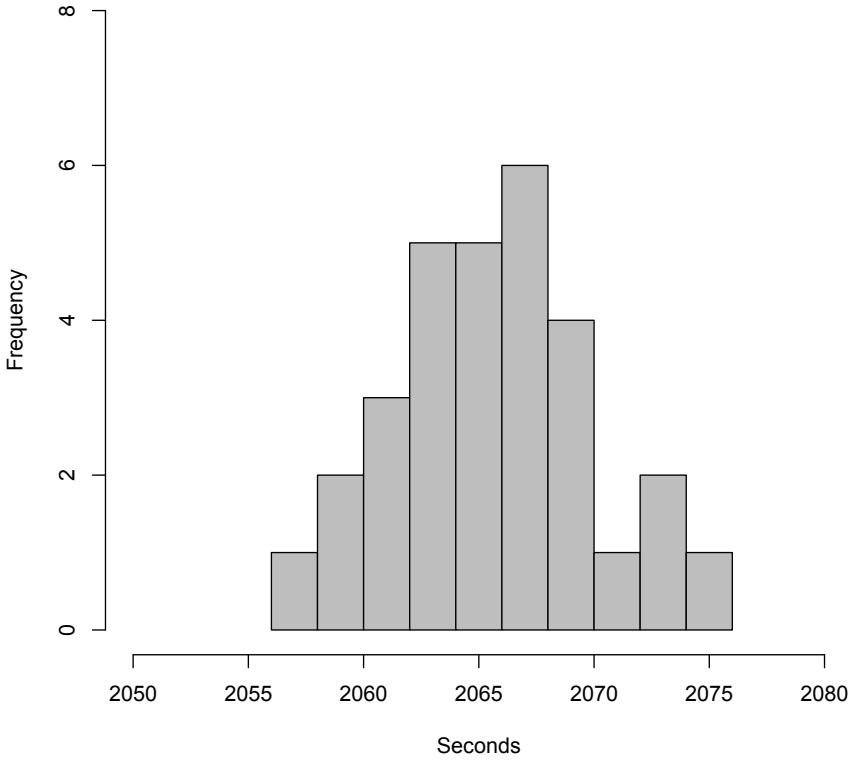

Figure 6.3: Sequential Migration Scenario with 16 VM's

As the diagram above shows, the function which was stated earlier works well. The results are gathered around 2065 seconds.

Dataset	**Mean** μ	**Variance** σ^2	**Standard Deviation** σ
16 Sequential	2065.20	17.33	4.16

Table 6.2: Statistical Overview of the 16 Sequential Migration Dataset

After the above 30 executions, which generally is not enough to make a clear

statement, it is now possible to state the following:

$$f(16) = 2065 + -10$$

From the above table 6.2, it is clear that the standard deviation has increased from earlier tests, which is obvious, since the three firsts test scenarios contained 50 conducted tests. The results shows that the variance is around 10 seconds more or less than 2065 seconds when migrating 16 virtual machines. With a 10 seconds variance, when the total execution time takes around 35 minutes, the results can be clarified as very precise. Using the above function, administrators will therefor have a clear indication of the time consumption before running such an execution.

6.1.2 Estimated Function on Parallel Migration Behavior

In the parallel scenario, generating a function is somewhat harder, since it is not that easy to conclude anything from the first three tests, other than it looks more towards an exponential growth. The following table gives an overview on the calculated mean for each dataset, from the results, with corresponding variance and standard deviation:

Dataset	**Mean μ**	**Variance σ^2**	**Standard Deviation σ**
2 Parallel	134.78	3.48	1.86
3 Parallel	189.92	6.44	2.53
4 Parallel	319.72	7.14	2.67

Table 6.3: Statistical Overview of the Parallel Migration Datasets

Some basic statistical analysis was done towards the parallel migration results as well. From the above table 6.3 it is clear that the standard deviation is slightly larger than for the sequential tests. Even so, for the first test 95% of the results were located in the area of *134.78sec +- 1.86sec*. For the second test the results are *189.92sec +- 2.53sec* and *319.72sec +- 2.67sec* for the last one. It is also worth noticing the increase in the standard deviation for the parallel scenario.

When looking at the three different parallel execution tests in comparison, it is hard to state any related function. The trend is although that the overhead, which comes from the saturation in the network channel, has a vital increase which may be exponential. To find out more about how the parallel migration trend is, it is necessary to also here perform a larger execution.

There will be used 16 virtual machines in this scenario as well, but not with any estimated function in advance, as this is very difficult when looking at the first three tests (2, 3 and 4 VM's).

Test results

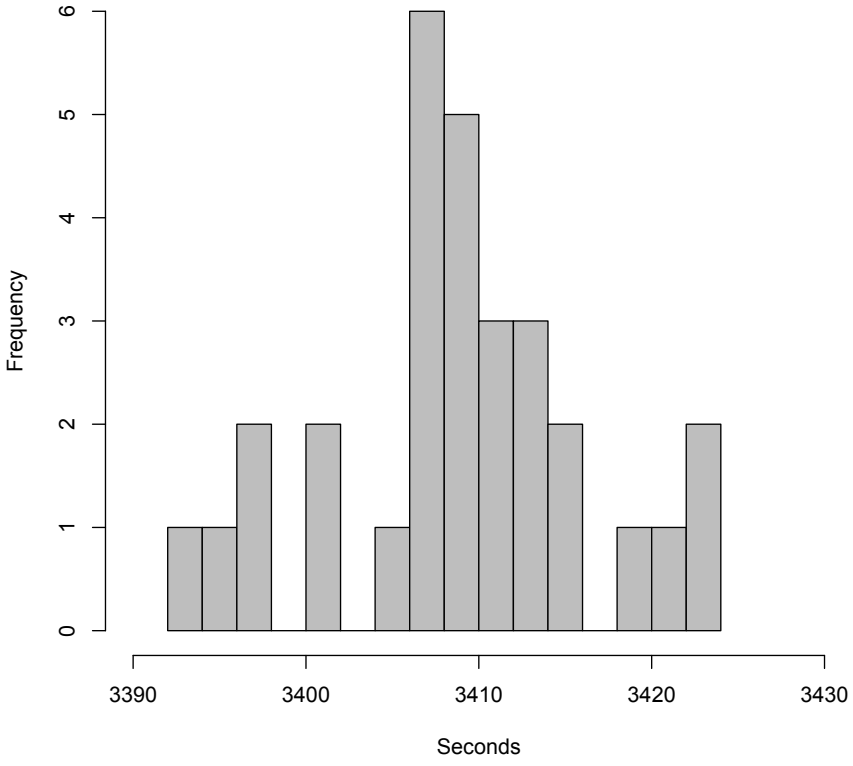

Figure 6.4: Parallel Migration Scenario with 16 VM's

The above diagram shows that for 16 VMs migrated in parallel it takes around 3410 seconds or in local time 56:50. Looking at the table below we can now see that the standard deviation has had a massive increase:

Dataset	**Mean** μ	**Variance** σ^2	**Standard Deviation** σ
16 Parallel	3408.33	55.74	7.46

Table 6.4: Statistical Overview of the 16 Parallel Migration Dataset

From this table the 16 VMs takes roughly *3408sec +- 7.5sec*. First of all, since the standard deviation is somewhat higher compared to previous numbers, it is a trend that running more and more VMs in parallel generates greater variance, which may end up in a less predictable result for the parallel scenario compared to the sequential scenario. Although this test was only conducted 30 times compared to previous 50.

From the results above the following estimated function was generated:

$$f(x) = 13x^2 + x + 90$$

The accuracy of the estimated formula is quite good as we can see from the table below:

Dataset	Mean	Estimated result	Deviation in %
2	134.78	144	-6.4%
3	189.92	210	-0.9%
4	319.72	302	0.5%
16	3408.33	3434	0.07%

Table 6.5: Statistical Overview of the Estimated Function Impact

As earlier thoughts mentioned the trend of the curve looked towards an exponential function, it is now clear that it follows the above quadratic function better. A percentage deviation of -6.4% is the largest deviation in the table above, while the rest is under 1%.

6.1.3 Sequential vs. Parallel Migration Conclusion

The above two sections, 6.1.1 and 6.1.2, discussed and went through the sequential and parallel scenario in a more detailed manner. From the above graphs it is now possible to conclude that the parallel migration is profitable for small scale scenarios, while sequential migration is profitable for large scale scenarios. The following table is a short summary on the different end-to-end times:

# VM's	Sequential	Parallel
2	243.90	134.78
3	374.92	189.92
4	504.58	319.72
16	2065.20	3408.33

Table 6.6: Statistical Overview of all Datasets

From the above numbers and some calculations on 8, 10 and 12 VMs using the estimated quadratic function for parallel migration, a graph was made to see the general curve of both the sequential and the parallel scenario:

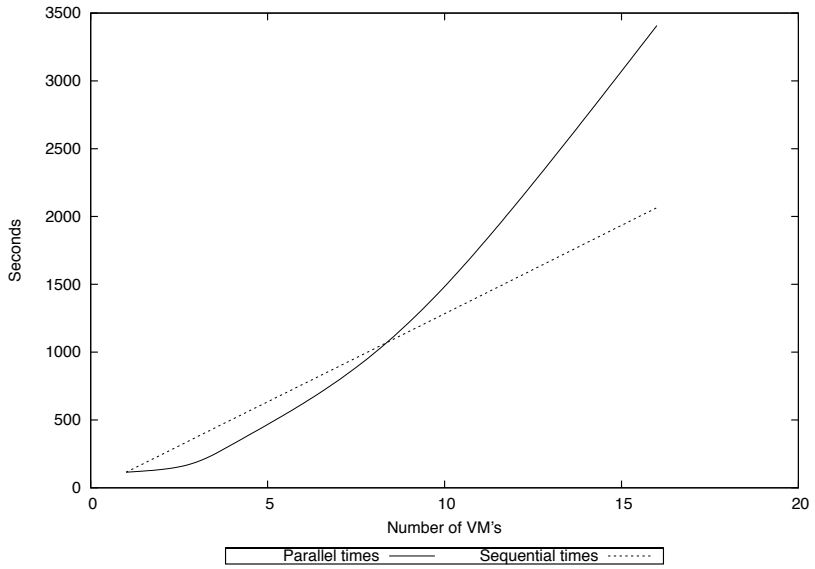

Figure 6.5: Result graph for Sequential and Parallel Migration Scenario

As shown by the graph above, when migrating 8-9 VMs in parallel, the sequential scenario starts to be beneficial. The curve of the parallel scenario follows from a quadratic function, while the sequential scenario is linear. Which means that it is possible to conclude that, by comparing these two scenarios, the sequential scenario is definitely more efficient in large scale scenarios and also not very far away from the parallel scenario in small scale as well.

The above scenarios are just two examples of how to migrate the virtual machines from one cloud to the other. A combination of this might result in a better outcome. For example when migrating 16 virtual machines, one could migrate in sequences with each sequence consisting of 2-3 VMs in parallel. This is something that may be taken up in future work.

6.2 Efficiency Theories

Throughout this thesis, the end-to-end has been the main focus. Stating which scenario that outperforms the other. So in previous chapters and sections the thesis has presented sequential and parallel migration scenarios as two options when migrating virtual machines. Both of the scenarios has been equal in terms of the environmental topology and the three stages is also generally equal. The following figure shows the three stages with corresponding down-time and end-to-end time:

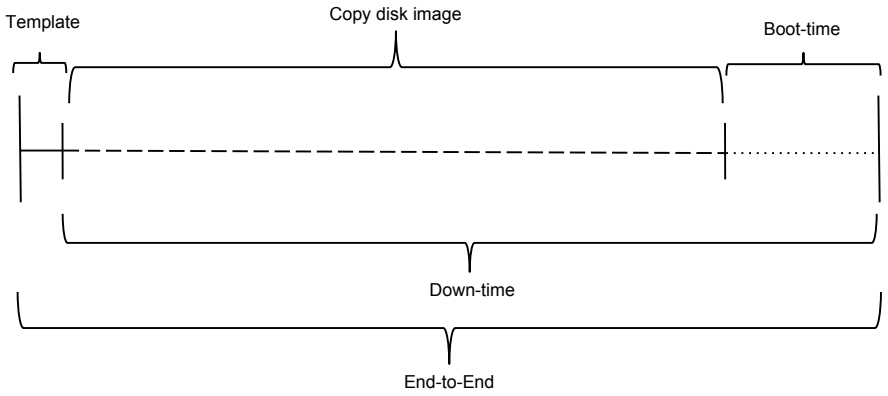

Figure 6.6: Flow of the Migration Process

The three stages are the following (which is the upper line in the figure above):

- Read/Write Template Stage
- Copy Disk Stage
- Boot Stage

The end-to-end time is the total time for the whole process to complete. The down-time is the time when the user, using the VM, don't have the ability to access the VM. In the above case, which is the the general one that have been discussed in this thesis, the down-time starts when the *Copy Disk Stage* starts. This means that the user do not have access to the VM for almost the same time as the total end-to-end time. This is generally not a desired scenario, as it will prevent users to do beneficial work for quite some time.

Taking a look at table 6.6 in section 6.1.3 in the parallel scenario for 16 virtual machines. All VM's will enter the *Copy Disk Stage* at the same time, which is, as mentioned earlier, when the down-time starts. On average, migrating 16 VM's in parallel takes 3408 seconds, which is close to 1 hour. This means that, if those 16 VM's had 16 distinct users, all those users could not do any work on those machines for almost 1 hour, since the *Read/Write Template Stage* takes only a few seconds and is therefor not included.

A general theory is therefor implemented, so that the down-time can be abbreviated, which again may result in a more efficient migration process.

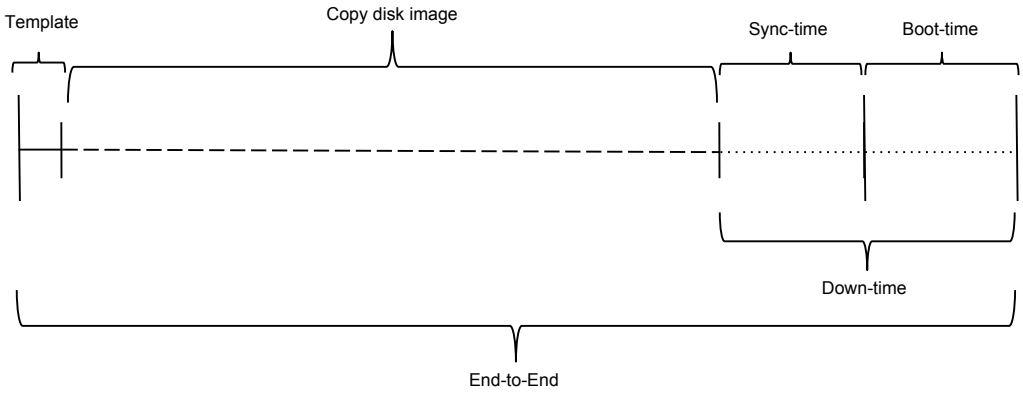

Figure 6.7: Flow of the Migration Process, with New Stage

Looking at the figure above, a new stage has been introduced.

- Read/Write Template Stage
- Copy Disk Stage
- Synchronize Blocks/Files Stage
- Boot Stage

First of all, the two first stages remains untouched and are still using the same amount of time as previous scenarios. Then comes the new introduced stage *Synchronize Blocks/Files Stage*. The idea of implementing this stage, is to prevent the down-time to start when the large *Copy Disk Stage* starts. The drawback when implementing this stage, is that it will affect the total end-to-end time, but when thinking thoroughly about it, the down-time is the important factor throughout the migration process.

In a more technical perspective, this new stage introduces a way to synchronize the disk at the VM after the copy is finished. If we look back at section 2.3, at page 19, a concept was discussed, called "dirty blocks". This is a modern stage in live migration, which synchronizes the changes that has been made towards the system after the copy was initiated. The same theory will be implemented here, but in this case for the disk image. This means that the shut-down of the VM can be postponed to after the *Copy Disk Stage*, and then initiated right before entering the *Synchronize Blocks/Files Stage*. This new stage will then synchronize the changes the user has been doing towards the disk while the *Copy Disk Stage* was processing.

The *Synchronize Blocks/Files Stage* will therefor vary in time in terms of how many

changes that has been done. Naturally, the more changes, the more data has to be synchronized when entering this stage. Although when comparing the above two figures 6.6 and 6.7, the idea is that the down-time will be shortened dramatically, unless the changes from the user is as large as the total disk image size that was copied. Which again will, in that case, double the total end-to-end time, even though this an unrealistic scenario.

There are some different ways of performing such an implementation and the next two sections 6.2.1 and 6.2.2 will discuss this new implementation in a more detailed manner.

6.2.1 "Dirty Blocks"

"Dirty Blocks" is a conceptual self-created theory from the "Dirty Pages" function in live migration, which was introduced in 2.3. The idea behind it is the same as described above, to copy the changes, afterwards, that was made towards the disk during the main copy stage. "Dirty Blocks" main focus is on the block changes on the disk image. There exists some already implemented disk synchronization tools on this field, which will soon be discussed. These tools did not have any intention to work in this manner, as they are solidly meant towards disk synchronization in general. In that respect, those tools will be used and tested with aim to fulfill and conclude my "Dirty Blocks" theory.

Data Generation

Before explaining and discussing the disk sync tools and theory, it is important to mention the data generation that will occur on the virtual machine as the *Copy Disk Stage* is processing. To see any affect at all using the "Dirty Blocks" theory, some changes needs to be done towards the disk. A script (disk-load.pl), which can be found in the Appendix, was therefor created, which only creates a new file on the disk and hammers the file with a certain amount of data.

disk-load.pl: Core code

```
1 my $filesize = $opts;
2 system("dd if=/dev/zero of=./load bs=1024 count=$filesize");
```

The user of the script specifies, in bytes, how large the new created file should be. In the following cases 100MB will be used.

Blocksync.py

The first disk synchronization tool is a python script called blocksync.py[2], which is developed by Robert Coup. The idea behind it is to copy the script on to the destination host and execute it with the following command:

blocksync.py: Execution example

```
1 sudo python blocksync.py /dev/source user@remotehost /dev/dest
```

When implementing blocksync.py into the main script, manager.pl, the core code needed to be changed, so that the copy stage was executed before the shut-down of the VM. A new method was also introduced:

manager.pl: Important new core code

```
1  @info_template = get_vminfo_from_cloud1();
2  my @mod_template = modify_template(@info_template);
3  create_template_on_cloud2();
4
5  copy_disk_from_cloud1_to_cloud2();
6  shutdown_vm_on_cloud1();
7
8  send_dirty_blocks();
9
10 create_vm_on_cloud2();
11 delete_vm_on_cloud1();
```

The *send_dirty_blocks()* method is now just an import of the blocksync.py script which is responsible for the synchronization of the disk images at the source and destination clouds.

As mentioned in section 5.2.1, the copy-stage and boot-stage together were tested, which showed that the approximate down-time for the user in those scenarios were calculated to be 115sec - 2sec (the template stage). The following graph shows the approximate down-time when using the "Dirty-blocks" theory with blocksync.pl as the main synchronization tool:

[2]https://gist.github.com/1338263

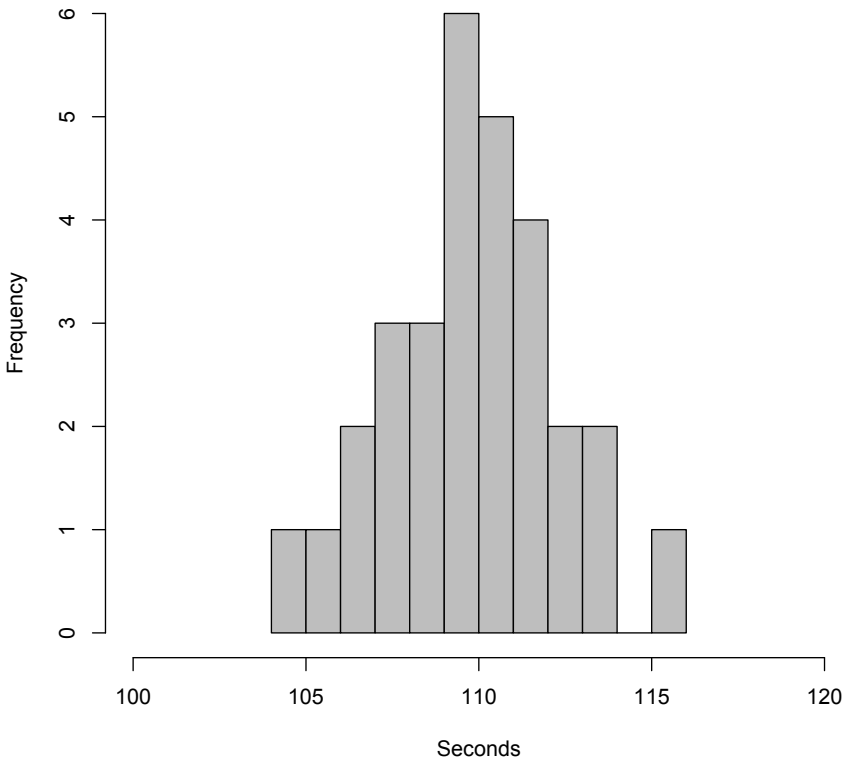

Figure 6.8: Down-time of one VM with blocksync.py

As shown by the above diagram, the down-time has not decreased at all, in fact ended up at the exact same numbers as earlier approaches, which are quite surprisingly results since the whole *Copy Disk Stage* is now replaced with the minor *Synchronize Blocks/Files Stage* of 100MB shown in figure 6.7.

The reason for these results is because the blocksync.pl needs to traverse the whole disk image an checksum each data on it before it can decide which data that has been changed during the copy stage. Since there are no metadata existing on blocks, just files, the checksum operation needs to be performed. In this environment the network speed approaches the disk I/O speed which makes this a pointless operation time-wise. The time for the sync-stage to checksum all the data and send the differences is about the same time it takes to send all the data all over again.

Even though this only increased the down-time by 2-3 seconds on average, it is still an idea that may have profitable larger impact in other environments. The execution environment in this case has high-speed network channels and slow running disks on single servers. If we imagine a scenario where there exists major datacenters, at both locations, that often has large disk-arrays, e.g. SAN or raid, and these two locations are geographically far away from each other, e.g. wan. In those cases the disk opera-

tion will have much greater I/O speeds than the network link between them, and it is thinkable that the solution using blocksync.py will create much more beneficial results on the user down-time.

Rsync

The next tool is *rsync*[3], which is a well known and common tool to use when synchronizing files or filesystems in general. As this thesis focuses on raw disk formats, *rsync* needs a patch to work on this specific image format. First of all one needs to compile *rsync-3.0.9*[4] from source, and also the *rsync-patches-3.0.9.tar.gz*[4] is needed. Inside this *tar.gz* package the *copy-devices.diff* patch is located which is needed for *rsync* to synchronize raw disk formats. This patch can be easily installed with the following commands:

rsync: Install copy-devices.diff patch

```
1 patch copy-devices.diff
2 ./configure
3 make
4 make install
```

Implementing the rsync functionality into the manager.pl script was simple. It was just to replace the old blocksync.pl script from the *send_dirty_blocks()* method with the following execution:

rsync: Execution example

```
1 rsync --checksum --perms --owner --group --sparse --partial --progress --copy-devices /
2         128.39.74.2:<image-path> 128.39.74.29:<image-path>
```

As the command above shows, the two image paths is where the supposed two different raw disk images on the separate clouds are located.

The following graph shows the approximate down-time when using the "Dirty-blocks" theory with rsync as the main synchronization tool:

[3]http://linux.die.net/man/1/rsync

[4]http://www.samba.org/ftp/rsync/src/

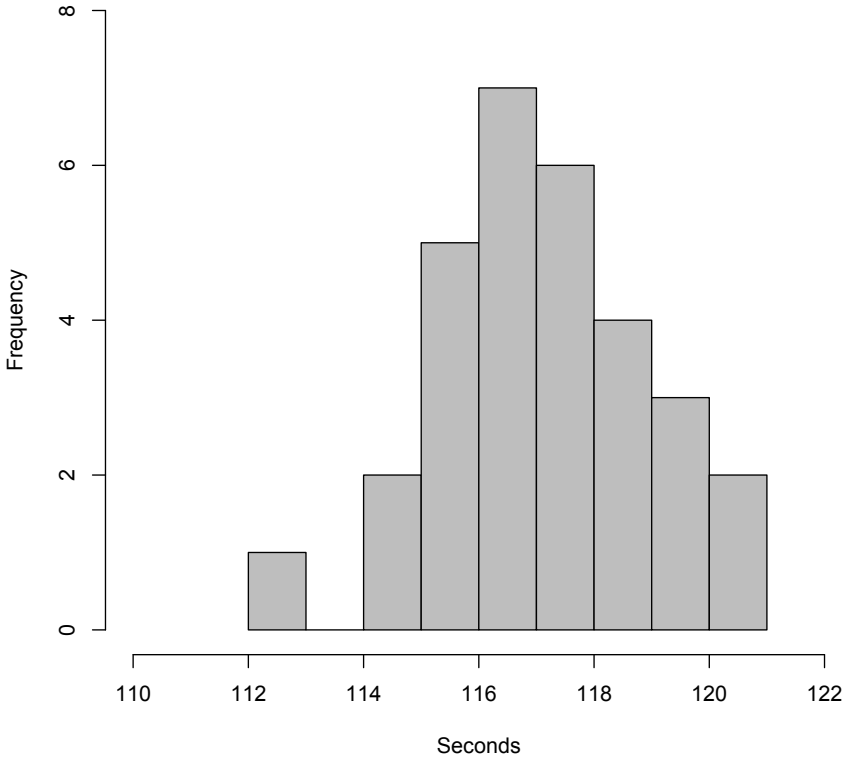

Figure 6.9: Down-time of one VM with rsync

The above graph shows that also this tool is not promising in the executed environment, due to the same problem as the *blocksync.py* script. The network outperforms the disk speed, which makes the checksum operation of the whole image file unnecessary time-wise. It is although difficult to conclude why this *rsync* solution is even slower on average than the previous *blocksync.py* script.

6.2.2 "Dirty Files"

The main reason why the above "Dirty-blocks" theory did not work properly in this thesis environment, was the disk I/O speed compared to the network speed. Checksum operations towards disk images are often very time consuming, which is not a profitable approach when the goal is to decrease the user down-time.

In light of this a new theory, aimed for this type of environment, which is self-created and called "Dirty Files", was developed. As the tests using the "Dirty Blocks" theory did not return satisfying numbers on the expected user down-time, a new approach had to be implemented. The theory from the "Dirty Pages" is still preserved, but the general difference is the avoidance of checksum operations towards the data blocks.

The main reason to develop a new theory at the *Synchronize Blocks/Files Stage* is to aim for the possibility to streamline the user down-time for this environment. Even though the previous theory did not work properly for this environment the main idea, which figure 6.7 shows, needs to be investigated further.

Technical specification

In the "Dirty Blocks" theory two already existing, disk synchronization, tools were presented. In this new theory *rsync* will be used as well, but in an own setup, which is self-created, to utilize the possibility to synchronize the changes made towards the disk.

manager.pl: Final core code

```
1  my @info_template = get_vminfo_from_cloud1();
2  my @mod_template = modify_template(@info_template);
3  create_template_on_cloud2();
4
5  copy_disk_from_cloud1_to_cloud2();
6  shutdown_vm_on_cloud1();
7
8  mount_disk_on_cloud1();
9  mount_disk_on_cloud2();
10
11 send_dirty_files();
12
13 umount_disk_on_cloud2();
14 umount_disk_on_cloud1();
15
16 create_vm_on_cloud2();
17 delete_vm_on_cloud1();
```

As the code above shows, the highlighted methods are the new ones that are introduced in this "Dirty Files" theory. What happens is that, first of all, the user still can perform tasks during the *copy_disk_from_cloud1_to_cloud2();*. Once the shut-down of the VM has been completed, the disk at cloud1 and cloud2 gets mounted.

mount_disk_on_cloud1(): Core code

```
1  my $PATH = "/var/lib/one/$VM_ID/images/disk.0";
2  my $MOUNT_PATH = "/media/$VM_ID";
```

mount_disk_on_cloud2(): Core code

```
1 my $PATH = "/var/lib/one/disks/$VM_ID/disk.0";
2 my $MOUNT_PATH = "/mnt/$VM_ID";
```

The disk image at cloud2 is the one that was copied, and therefor without any changes. The disk image at cloud1 may, although, have had some changes during the copy stage.

Once both of the images are mounted, the *send_dirty_files()* method begins.

send_dirty_files(): Core code

```
1 my $cmd ="rsync -avPS /media/$VM_ID/ root@" . "$CLOUD2_IP:/mnt/";
2 my $ans = $ssh_cloud1_root->exec($cmd);
```

The *send_dirty_files()* method will then use *rsync* to synchronize the two mounted filesystems on cloud1 and cloud2.

Results

As described in section 5.2.1, the boot time for the used image was tested be consistent around 20 seconds. While the template stage was around 2-3 seconds. Meaning that the rest of the end-to-end time was the copy stage time which proved to be consistent around 90-95 seconds. This led to an approximate down of 110-112 seconds (Copy Stage + Boot Stage - Template Stage). The following table shows an overview on the expected and predictive copy times for different sized disk changes (all numbers are approximations and based on the Copy Stage of the 2GB image):

Disk Changes	Expected Copy Time
2GB	92.5sec
1GB	46sec
500MB	23sec
100MB	4.5sec

Table 6.7: Expected Copy Times

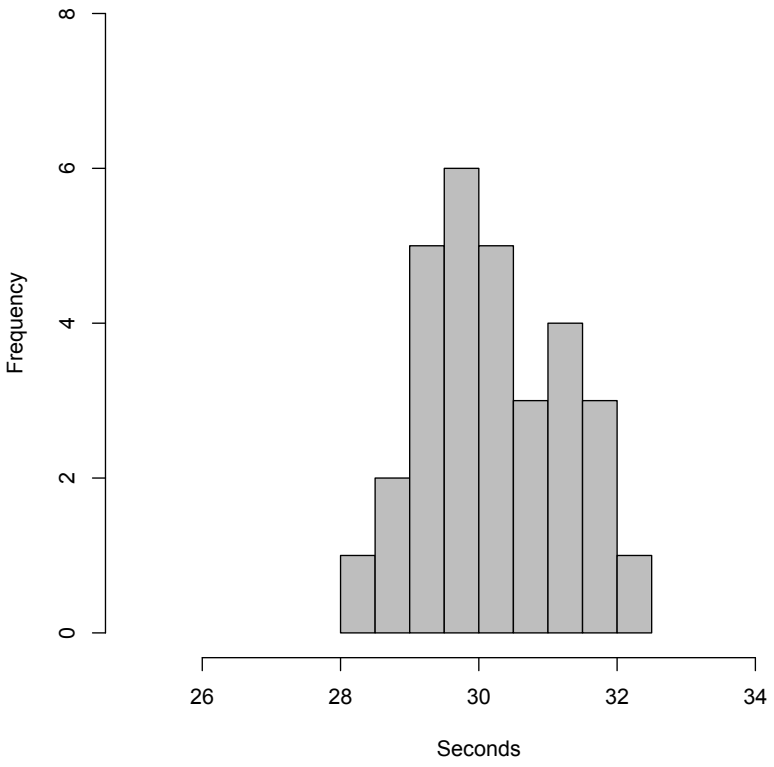

Figure 6.10: Down-time of one VM with "Dirty Files" and 100MB of disk changes

From this graph it becomes clear that the down-time has decreased dramatically. Based on the above numbers, the boot-time of the VM remains consistent, around 20 seconds. The expected copy time for 100MB was 4.5 seconds. Due to some time consumption when mounting and umounting filesystems, the final down-time ended up around 30 seconds. Which means that in this case the *Copy Disk Stage* which takes around 90-95 seconds has in this case been replaced with a *Synchronize Blocks/Files Stage* of approximately 10 seconds.

In cases where a VM, for example, hosts a database, a lot of changes could be made towards the disk during the *Copy Disk Stage*. To see wether the "Dirty Files" theory remains consistent and follows the expectation table above, 6.7, a disk change of 500MB will now be analyzed.

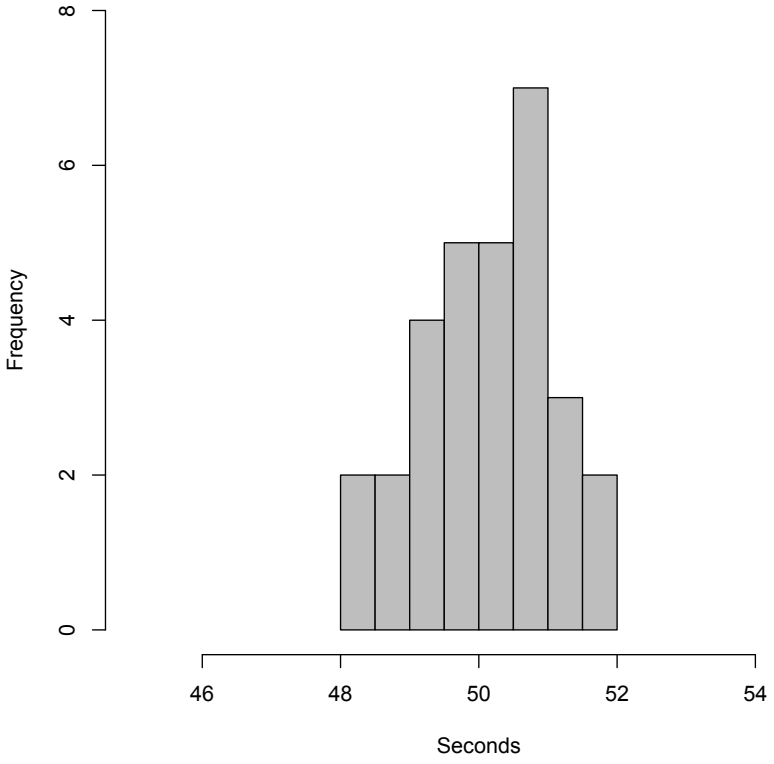

Figure 6.11: Down-time of one VM with "Dirty Files" and 500MB of disk changes

In table 6.7, 500MB should take approximately 23 seconds to copy. Looking at the above graph this number makes sense when adding the expected boot-time of 20 seconds and also the remaining 5 seconds of overhead due to mounting and umounting the filesystems. Which makes the calculated down-time for such a scenario to end up around 50 seconds.

Conclusion

The two different efficiency theories discussed here, "Dirty Blocks" and "Dirty Files", both have the potential to decrease the overall migration down-time. The "Dirty-Blocks" theory will most probably work in more realistic environments, which was not tested here, and the "Dirty Files" which showed a dramatic decrease for this specific environment.

Both ideas are meant to pursue the thoughts on investigating such a theory even further, where the down-time can be even more decreased to bring this solution towards production and realistic scenarios. Further efficiency possibilities will be taken up in section 7.2 at page 74.

Chapter 7

Conclusion

The prototype developed throughout this project opens up a lot of ideas and problems that exists across different cloud vendors today. To draw a line back to the beginning of the project, the problem statement was as follows:

How can we move virtual machines across heterogeneous cloud environments without common dependencies, like shared storage, in order to gain flexibility and prevent vendor lock-in?

Hosting multiple yet different private clouds or clouds with different virtualization technology at the underlying layer, is always attractive as different vendors aims to attract users by releasing product to perform well at certain tasks. The possibility to have the freedom to choose among them and freely configure them to suite the whole infrastructure is what flexibility really is. In this project a tool was successfully developed to gain this flexibility in an infrastructure environment, or in cases where there is a need to deploy an infrastructure out to someone else. Having this flexibility to choose among different hypervisors and cloud vendors circumvents the vendor lock-in aspect.

As mentioned earlier, a solution focusing on migrating virtual machines to other clouds or platforms can also create flexibility for businesses on when to perform the migration process. There exists numerous policies which may be trivial towards a choice to perform migration of business infrastructures. One can choose to migrate on behalf of cost effectiveness or power consumption. In a hybrid solution there are simple approaches to look at which public cloud that has the most favorable pricing scheme, which it will be easy to migrate among them. Electricity prices may have a lot of variation from one platform or even country. It would then be beneficial to migrate to the one which has the cheapest electricity supply. All these policy decision makings will then be possible without any reconfiguration or installment, which will maintain the operation and portability.

For services that genuinely needs, for example, 99% uptime a solution like the one in this thesis is probably not the best idea, since live migration is more suited for less down-time. Although there are scenarios where this solution also solves the issue with

99% uptime. Webfarm services are typical examples where there is a need for as good uptime as possible. During maintenance of such a service, it is a wise choice to migrate the infrastructure on to another physical host or datacenter. Instead of migrating the whole infrastructure, which in this case would lead to some down-time (boot-time + overhead), one can, for example, divide the resources and migrate 50% at a time. It would of course lead to less resources to handle the amount of traffic/webrequests that is progressing during the migration process, but it would still solve the uptime policy of 99%.

As this project is only a prototype, aiming to experiment a way to move VMs across heterogeneous clouds, there exists some drawbacks. Especially when executing in other, more realistic, environments e.g. across multiple networks, borders or even across major geographic locations. In those scenarios the following drawbacks could potentially occur:

- **Lack of Control:**
  When migrating virtual machines across networks, there are always a chance that something goes wrong, which is more or less out of control or reach for the administrator. This is also one of the main reasons why the project, due to potential time constraints were implemented in small scale network.
- **Time Consuming:**
  As far as the prototype has been developed, the migration process could be seen as time consuming. Although locally in this project environment it works fast, and could have worked even faster by changing some parameters like disk and network switch, but those parameters were, with intention, left out to generate more realistic time scenarios.
- **Virtual Machine Down-Time:**
  Another drawback is of course the virtual machine down time. That is the general idea on why the live migration concept was developed in the beginning. This project had never any intention of getting close to the down-time results using live migration, but to observe and streamline the possibilities when using cold migration to succeed on migrating virtual machines across heterogeneous clouds.

The results gathered in the project opens up the discussion on cold migration vs live migration. There exists a considerable amount of down-time using cold migration compared to live migration, which was well known before the start of the project, and was never any intention to solve as well. As far as the research on this field has come, live migration is not possible across heterogeneous cloud environments without shared storage. Therefor was this thesis a proposal and prototype to the idea of pursuing this thought in the field towards live migration.

The first results gathered in the "Results" chapter, were quite convincing in terms of reliability. Of course this had to do with the short travel distance between the two clouds, but still it strengthens the conclusion. It was also vital to look at some different

migration scenarios, as there will be multiple ways of executing the migration process. A conclusion could be easily drawn towards sequential migration for large scale migration scenarios compared to parallel migration for small scale scenarios. As was mentioned in section 6.2.2, further scenarios would be a nice feature in future research, with even more predictive algorithms to conclude the benefits.

A start towards the efficiency on the migration down-time was also necessary to draw, which will open up for future research on this field.

7.1 Further Development

Since the developed tool in this project is so far only at prototype level, several ideas are unsolved, but up for further development to bring the tool forward towards production ready.

The first idea is plugin support. As this project have mainly concentrated on using *OpenNebula* as the cloud provider, but with different hypervisor at the bottom layer, it as a good idea to create support for other private cloud providers such as *OpenStack* and *VMware*. The main difference is at the template stage. Therefor a plugin support seems reasonable by presenting a user option where one can choose which cloud provider that is hosting the destination cloud. A typical execution example will be as follows:

manager.pl: Future execution example with plugin support

```
./manager.pl -v -d -s [source-ip] -D [dest-ip] -p [source-pw] -P [dest-pw] /
             -u [source-user] -U [dest-user] -i [vm-id] -f [cloud plugin] -o [logfile]
```

The only difference would be to send possible options in to the new *-f flag*; *opennebula*, *openstack* and *vmware*. Each plugin will hold the basic template characters for all the cloud providers.

A thought which can be further developed is also public cloud support. This creates hybrid cloud possibilities which are truly beneficial performance wise since public providers have major datacenters with huge computational capacity. One can therefor, for example, create an EC2 plugin which will support deployment to an Amazon EC2 cloud.

The solution so far has not had any focus on the networking aspect. Throughout the project all machines have used public ip's to ease the access. So there is some work to do to get private networks to work from source to destination.

Since it is only a prototype and also due to time constraints the general testing and robustness of the code has not been done thoroughly. This is probably the first operation that should be solved, to create a solid base, before developing the tool even further.

7.2 Future Work

Throughout the "Analysis and Discussion" chapter some future work was proposed, and in the above section some new ideas to the implemented prototype were addressed.

Taking a look back at the flow of the project, the following list summarizes the two factors of the project that may have been done differently:

- **Availability:**
  The main drawback at the start of the project were, as mentioned earlier in section 2.2.6, the OpenStack error which led to service unavailability. In those cases, time was spent to reconfigure and reinstall the base fundamentals.
- **Experimental study:**
  Due to the first concern, some time was spent on a technical and experimental study to create a robust cloud (although necessary) which would last throughout the project. Since the focus of this thesis was on open-source software, a general lack of documentation and error reports were a fact, which extended the scheduled time for this study operation to be conducted.

Today there exists some great thoughts and ideas on how to efficiently migrate virtual machines across different virtualization platforms [24, 25, 26, 27, 28]. They are although not focusing on the cloud level, but the function of these works may be taken into consideration when doing further work on this project. There are also some efficiency algorithms [29, 30] which may be implemented to pursue the migration thought even further towards live migration.

The previous mentioned earlier work did not relay on the cloud level, which is the main goal of this thesis. There are however some previous work that has been doing it at this level as well [31], but this has not been focusing on the heterogeneous environments in terms of having no common dependencies, like shared storage etc. Although this work could be very interesting to take into account when doing further research.

As described in the "Further Development" section above, the prototype has not been focusing on the networking aspect across the different clouds. Some new features has now been matured in this area and been taken up in some other work [32, 33]. This has its main focus on using the *OpenFlow* protocol which enables the possibility of migrating datacenters across different platforms. Along with the *Open vSwitch* project [34, 35], these two upcoming softwares creates new opportunities to set up and manage network across virtualization and cloud platforms.

Bibliography

[1] Hugh Dingle. *Migration: the biology of life on the move*. Oxford University Press, 1996.

[2] Paul Ruth, Xuxian Jiang, Dongyan Xu, and Sebastien Goasguen. Virtual distributed environments in a shared infrastructure. *Computer*, 38(5):63–69, 2005.

[3] Pengcheng Liu, Ziye Yang, Xiang Song, Yixun Zhou, Haibo Chen, and Binyu Zang. Heterogeneous live migration of virtual machines. In *In International Workshop on Virtualization Technology (IWVT'08)*, 2008.

[4] Daniel A. Menascé. Virtualization: concepts, applications, and performance modeling. In *Proceedings of 31st International Computer Measurement Group Conference*, Orlando, Florida, USA, December 2005.

[5] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412 Ð421, 1974.

[6] Neil McAllister. Server virtualization. **http://www.infoworld.com/d/virtualization/server-virtualization-under-hood-147**, February 12th 2007.

[7] George Ou. Introduction to server virtualization. **http://articles.techrepublic.com.com/5100-10879_11-6074941.html**, May 22nd 2006.

[8] Tim Abels, Puneet Dhawan, and Balasubramanian Chandrasekaran. An overview of xen virtualization. *Dell Power Solutions*, August 2005.

[9] Guangda Lai, Hua Song, and Xiaola Lin. A service based lightweight desktop virtualization system. In *2010 International Conference on Service Sciences*, 2010.

[10] Zhang Qiang, Wu Yunlong, Cui Dong, and Dang Zhuang. Research on the security of storage virtualization based on trusted computing. In *2010 International Conference on Networking and Digital Society*, 2010.

[11] Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of SC '08 ACM/IEEE conference on Supercomputing*, 2008.

[12] N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba. Network virtualization: State of the art and research challenges. *Communications Magazine, IEEE*, July 2009.

[13] Tathagata Das, Pradeep Padala, Venkata N. Padmanabhan, Ramachandran Ramjee, and Kang G. Shin. Litegreen: saving energy in networked desktops using virtualization. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, Berkeley, CA, USA, 2010.

[14] Mohd Khairil and Mohd Appandi. Comparison of virtual machines using different virtualization technologies on linux platform, May 2006.

[15] Chen Xianqin, Wan Han, Wang Sumei, and Long Xiang. Seamless virtual machine live migration on network security enhanced hypervisor. In *Broadband Network & Multimedia Technology. IC-BNMT '09. 2nd IEEE International Conference*, 2009.

[16] Peter Mell and Tim Grance. Effectively and securely using the cloud computing paradigm. *NIST, Information Technology Laboratory*, August 2009.

[17] Laura Savu. Cloud computing: Deployment models, delivery models, risks and research challenges. In *Proceedings of the Computer and Management (CAMAN), 2011 International Conference*, Wuhan, 2011.

[18] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *CloudCom '09 Proceedings of the 1st International Conference on Cloud Computing*, Beijing, China, December 2009.

[19] Opennebula, last stable release. **http://www.opennebula.org/software:software**.

[20] Opennebula, installing the software. **http://www.opennebula.org/documentation:rel3.4:ignc**.

[21] Opennebula, installing sunstone front-end. Website. **http://www.opennebula.org/documentation:rel3.4:sunstone**.

[22] Xen, installing the software. **http://xen.org/support/documentation.html**.

[23] Nfs, installing the software. **http://tldp.org/HOWTO/NFS-HOWTO/server.html**.

[24] Michael Nelson, Beng hong Lim, and Greg Hutchins. Fast transparent migration for virtual machines. In *Proceedings of the annual conference on USENIX Annual Technical Conference*. USENIX Association, 2005.

[25] Franco Travostino, Paul Daspit, Leon Gommans, Chetan Jog, Cees de Laat, Joe Mambretti, Inder Monga, Bas van Oudenaarde, Satish Raghunath, and Phil Yonghui Wang. Seamless live migration of virtual machines over the man/wan. *Future Gener. Comput. Syst.*, 22(8):901–907, October 2006.

[26] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schioberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd international conference on Virtual execution environments (New*, pages 169–179. ACM Press, 2007.

[27] K. K. Ramakrishnan, Prashant Shenoy, and Jacobus Van Der Merwe. Live data center migration across wans: A robust cooperative context aware approach. submitted for publication, 2007.

[28] Yingwei Luo, Binbin Zhang, Xiaolin Wang, Zhenlin Wang, Yifeng Sun, and Haogang Chen. Live and incremental whole-system migration of virtual machines using block-bitmap. In *2008 IEEE International Conference on Cluster Computing (Cluster 2008)*, pages 99–106, September 2008.

[29] Hsu Mon Kyi and Thinn Thu Naing. An efficient approach for virtual machines scheduling on a private cloud environment. In *2011 4th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, pages 365–369, October 2011.

[30] Katharina Haselhorst, Matthias Schmidt, Roland Schwarzkopf, Niels Fallenbeck, and Bernd Freisleben. Efficient storage synchronization for live migration in cloud infrastructures. In *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2011)*, pages 511–518, February 2011.

[31] Takahiro Hirofuchi, Hirotaka Ogawa, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. A live storage migration mechanism over wan for relocatable virtual machine services on clouds. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 460–465, Washington, DC, USA, 2009. IEEE Computer Society.

[32] Fang Hao, T. V. Lakshman, Sarit Mukherjee, and Haoyu Song. Enhancing dynamic cloud-based services using network virtualization. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, VISA '09, pages 37–44, New York, NY, USA, 2009. ACM.

[33] Timothy Wood, K. K. Ramakrishnan, Prashant Shenoy, and Jacobus van der Merwe. Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, pages 121–132, New York, NY, USA, 2011. ACM.

[34] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, and Scott Shenker. Extending networking into the virtualization layer. In *8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII).New YorkCity,NY(October 2009*, 2009.

[35] Yan Pu, Yilong Deng, and A. Nakao. Cloud rack: Enhanced virtual topology migration approach with open vswitch. In *2011 International Conference on Information Networking, ICOIN 2011*, pages 160–164, 2011.

Appendix A

manager.pl

The manager.pl script

```
#! /usr/bin/perl -w

# Needed packages
use Getopt::Std;
use strict "vars";
use Net::SSH::Expect;
use Net::SCP::Expect;
use POSIX qw(strftime);

# Global vars
my $VERBOSE = 0;
my $DEBUG = 0;

# Handle flags and args
# Example: c == "-c", c: == "-c argument"
my $opt_string = "vdhs:D:p:P:u:U:i:o:";
getopts("$opt_string",\my %opt ) or usage() and exit 1;

# Print help msg if -h is invoked
if ( $opt{h} ){
        usage();
        exit 0;
}
# Handle other user input
$VERBOSE = 1 if $opt{v};
$DEBUG = 1 if $opt{d};

my $CLOUD1_IP = $opt{'s'};
my $CLOUD2_IP = $opt{'D'};
my $CLOUD1_PW = $opt{'p'};
my $CLOUD2_PW = $opt{'P'};
my $CLOUD1_USER = $opt{'u'};
my $CLOUD2_USER = $opt{'U'};
my $VM_ID = $opt{'i'};
my $LOGFILE = $opt{'o'};

#---- Start: Main script content ----#

verbose("Verbose is enabled\n");
debug("Debug is enabled\n");

#clears logfile
#open(FILE, ">$LOGFILE");
#print FILE "";
#close(FILE);

my $ssh_cloud1 = Net::SSH::Expect->new (
```

```
48          host => $CLOUD1_IP,
49          password => $CLOUD1_PW,
50          user => $CLOUD1_USER,
51          raw_pty => 1,
52          timeout => 3
53      );
54      $ssh_cloud1->login();
55      verbose("Successfully logged into cloud1.....\n");
56
57      my $ssh_cloud1_root = Net::SSH::Expect->new (
58          host => $CLOUD1_IP,
59          password => $CLOUD1_PW,
60          user => "root",
61          raw_pty => 1,
62          timeout => 3
63      );
64      $ssh_cloud1_root->login();
65      verbose("Successfully logged into cloud1 as root.....\n");
66
67      my $ssh_cloud2 = Net::SSH::Expect->new (
68          host => $CLOUD2_IP,
69          password => $CLOUD2_PW,
70          user => $CLOUD2_USER,
71          raw_pty => 1,
72          timeout => 3
73      );
74      $ssh_cloud2->login();
75      verbose("Successfully logged into cloud2.....\n");
76
77      my $ssh_cloud2_root = Net::SSH::Expect->new (
78          host => $CLOUD2_IP,
79          password => $CLOUD2_PW,
80          user => "root",
81          raw_pty => 1,
82          timeout => 3
83      );
84      $ssh_cloud2_root->login();
85      verbose("Successfully logged into cloud2 as root.....\n");
86
87      my @info_template = get_vminfo_from_cloud1();
88      my @mod_template = modify_template(@info_template);
89      create_template_on_cloud2();
90
91      copy_disk_from_cloud1_to_cloud2();
92      shutdown_vm_on_cloud1();
93
94      mount_disk_on_cloud1();
95      mount_disk_on_cloud2();
96
97      send_dirty_files();
98
99      umount_disk_on_cloud2();
100     umount_disk_on_cloud1();
101
102     create_vm_on_cloud2();
103     delete_vm_on_cloud1();
104
105     #---- End: Main script content ----#
106     sub out {
107         open(OUT, ">>$LOGFILE") or die "Can't write to file '$LOGFILE' [$!]\n";
108         print OUT $_[0];
109         close(OUT);
110     }
111
112     sub mount_disk_on_cloud1 {
113         my $PATH = "/var/lib/one/$VM_ID/images/disk.0";
```

```
114         my $MOUNT_PATH = "/media/$VM_ID";
115
116         verbose("Mounting disk on cloud1: ");
117         $ssh_cloud1_root->exec("mkdir $MOUNT_PATH");
118         $ssh_cloud1_root->exec("losetup /dev/loop0 $PATH");
119         $ssh_cloud1_root->exec("kpartx -a -v /dev/loop0");
120         $ssh_cloud1_root->exec("mount /dev/mapper/loop0p1 $MOUNT_PATH");
121         verbose("Done!\n");
122     }
123
124     sub mount_disk_on_cloud2 {
125         my $PATH = "/var/lib/one/disks/$VM_ID/disk.0";
126         my $MOUNT_PATH = "/mnt/$VM_ID";
127
128         verbose("Mounting disk on cloud2: ");
129         $ssh_cloud2_root->exec("mkdir $MOUNT_PATH");
130         $ssh_cloud2_root->exec("losetup /dev/loop1 $PATH");
131         $ssh_cloud2_root->exec("kpartx -a -v /dev/loop1");
132         $ssh_cloud2_root->exec("mount /dev/mapper/loop1p1 $MOUNT_PATH");
133         verbose("Done!\n");
134     }
135
136     sub send_dirty_files {
137         verbose("Starts rsync VM $VM_ID disk from cloud1 to cloud2.....\n");
138         my $cmd ="rsync -avPS /media/$VM_ID/ root@" . "$CLOUD2_IP:/mnt/";
139         my $ans = $ssh_cloud1_root->exec($cmd);
140
141         my $copy = 1;
142         while($copy) {
143             my $psaux = $ssh_cloud1_root->send("ps aux | grep \"rsync -azvv\"");
144             if($psaux =~ /.*rsync -azvv.*/) {
145                 $copy = 1;
146             }
147             else {
148                 $copy = 0;
149             }
150             verbose("Disks are still syncing...\n");
151             sleep(1);
152         }
153
154         print "$ans\n";
155         verbose("Sync is DONE!!!\n");
156     }
157
158     sub umount_disk_on_cloud1 {
159         my $PATH = "/var/lib/one/$VM_ID/images/disk.0";
160         my $MOUNT_PATH = "/media/$VM_ID";
161
162         verbose("Umounting disk on cloud1: ");
163         $ssh_cloud2_root->exec("umount $MOUNT_PATH");
164         $ssh_cloud2_root->exec("dmsetup remove_all ");
165         $ssh_cloud2_root->exec("losetup -d /dev/loop0");
166         $ssh_cloud2_root->exec("rm -r $MOUNT_PATH");
167         verbose("Done!\n");
168     }
169
170     sub umount_disk_on_cloud2 {
171         my $PATH = "/var/lib/one/disks/$VM_ID/disk.0";
172         my $MOUNT_PATH = "/mnt/$VM_ID";
173
174         verbose("Umounting disk on cloud2: ");
175         $ssh_cloud2_root->exec("umount $MOUNT_PATH");
176         $ssh_cloud2_root->exec("dmsetup remove_all ");
177         $ssh_cloud2_root->exec("losetup -d /dev/loop1");
178         $ssh_cloud2_root->exec("rm -r $MOUNT_PATH");
179         verbose("Done!\n");
```

```
180
181 }
182
183 sub get_vminfo_from_cloud1 {
184         my $vminfo = $ssh_cloud1->exec("onevm show $VM_ID");
185         (my $a, my $b) = split("VIRTUAL MACHINE TEMPLATE", $vminfo);
186         (my $c, my $d) = split("VIRTUAL MACHINE HISTORY", $b);
187         my @template =  split("\n", $c);
188         verbose("Gets template info from VM $VM_ID on cloud1.....\n");
189         return @template;
190 }
191
192 sub get_vminfo_from_cloud2 {
193         my $vminfo = $ssh_cloud2->exec("onevm show $VM_ID");
194         (my $a, my $b) = split("VIRTUAL MACHINE TEMPLATE", $vminfo);
195         (my $c, my $d) = split("VIRTUAL MACHINE HISTORY", $b);
196         my @template =  split("\n", $c);
197         verbose("Gets template info from VM $VM_ID on cloud2.....\n");
198         return @template;
199 }
200
201 sub modify_template {
202         my @template = {};
203
204         for (my $i = 0; $i < (scalar(@_) - 2); $i++) {
205                 if ($_[$i] =~ /VMID=.*/) {
206                         next;
207                 }
208                 $template[$i] = $_[$i];
209         }
210
211         my $count_nic = 0;
212         for (my $i = 0; $i < (scalar(@template) - 1); $i++) {
213                 if ($template[$i] =~ /NIC=.*/) {
214                         while(!($template[$i] =~ /.*\]/)) {
215                                 $count_nic++;
216                                 $i++;
217                         }
218                 }
219         }
220
221         for (my $i = 0; $i < (scalar(@template) - 1); $i++) {
222                 if ($template[$i] =~ /NIC=.*/) {
223                         splice(@template, $i, $count_nic + 1);
224                 }
225         }
226
227         my $count_disk = 0;
228         for (my $i = 0; $i < (scalar(@template) - 1); $i++) {
229                 if ($template[$i] =~ /DISK=.*/) {
230                         while(!($template[$i] =~ /.*\]/)) {
231                                 $count_disk++;
232                                 $i++;
233                         }
234                 }
235         }
236
237         for (my $i = 0; $i < (scalar(@template) - 1); $i++) {
238                 if ($template[$i] =~ /DISK=.*/) {
239                         splice(@template, $i, $count_disk + 1);
240                 }
241         }
242
243
244
245
```

```
246         my @NIC = ("NIC=[", "  NETWORK=Simple,", "  NETWORK_UNAME=oneadmin ]");
247         my @DISK = ("DISK=[", "  BUS=ide,", "  CLONE=yes,", "  DRIVER=raw,",
248                     "  READONLY=no,", "  SAVE=no,", "  SOURCE=/var/lib/one/disks/$VM_ID/disk.0,",
249                     "  TARGET=sda,", "  TYPE=disk ]");
250         my @mod_template = (@template, @NIC, @DISK);
251
252         verbose("Successfully modified template.....\n");
253
254         return @mod_template;
255     }
256
257     sub shutdown_vm_on_cloud1 {
258         verbose("VM $VM_ID goes down on cloud1.....\n");
259         $ssh_cloud1->exec("onevm shutdown $VM_ID");
260         sleep(1);
261         verbose("Successfully shutted down VM $VM_ID on cloud1.....\n");
262     }
263
264     sub shutdown_vm_on_cloud2 {
265         verbose("VM $VM_ID goes down on cloud2.....\n");
266         $ssh_cloud2->exec("onevm shutdown $VM_ID");
267         sleep(1);
268         verbose("Successfully shutted down VM $VM_ID on cloud2.....\n");
269     }
270
271     sub delete_vm_on_cloud1 {
272         verbose("Deleting VM $VM_ID on cloud1.....\n");
273         $ssh_cloud1->exec("onevm delete $VM_ID");
274         verbose("VM $VM_ID successfully deleted.....\n");
275     }
276
277     sub delete_vm_on_cloud2 {
278         verbose("Deleting VM $VM_ID on cloud1.....\n");
279         $ssh_cloud2->exec("onevm delete $VM_ID");
280         verbose("VM $VM_ID successfully deleted.....\n");
281     }
282
283     sub copy_disk_from_cloud1_to_cloud2 {
284         my $disk_info = $ssh_cloud1->exec("ls -l $VM_ID/images/disk.0");
285         my $bytes = 0;
286         if ($disk_info =~ /oneadmin oneadmin (\d*) /) {
287             $bytes = $1;
288         }
289
290         $ssh_cloud1->exec("mkdir disks/$VM_ID");
291
292         verbose("Starts copying VM $VM_ID disk image from cloud1 to cloud2.....\n");
293         $ssh_cloud1->send("scp $VM_ID/images/disk.0 /
294                                     $CLOUD2_USER" . "@" . "$CLOUD2_IP:disks/$VM_ID");
295         $ssh_cloud1->waitfor('^.*$', 1) or die "prompt 'password' not found after 1 second\n";
296         $ssh_cloud1->send("$CLOUD2_PW");
297
298         my $bytes2 = 0;
299         while ($bytes != $bytes2) {
300             sleep(2);
301             $disk_info = $ssh_cloud2->exec("ls -l disks/$VM_ID | grep disk.0");
302             if ($disk_info =~ /oneadmin oneadmin (\d*) /) {
303                 $bytes2 = $1;
304             }
305             verbose("$bytes <-----> $bytes2\n");
306         }
307         verbose("Successfully disk copy.....\n");
308     }
309
310
311
```

```
sub copy_disk_from_cloud2_to_cloud1 {
	my $disk_info = $ssh_cloud2->exec("ls -l $VM_ID/images/disk.0");
	my $bytes = 0;
	if ($disk_info =~ /oneadmin oneadmin (\d*) /) {
		$bytes = $1;
	}

	$ssh_cloud2->exec("mkdir disks/$VM_ID");

	verbose("Starts copying VM $VM_ID disk image from cloud2 to cloud1.....\n");
	$ssh_cloud2->send("scp $VM_ID/images/disk.0 /
				$CLOUD1_USER" . "@" . "$CLOUD1_IP:disks/$VM_ID");
	$ssh_cloud2->waitfor('^.*$', 1) or die "prompt 'password' not found after 1 second\n";
	$ssh_cloud2->send("$CLOUD1_PW");

	my $bytes2 = 0;
	while ($bytes != $bytes2) {
		sleep(2);
		$disk_info = $ssh_cloud1->exec("ls -l disks/$VM_ID | grep disk.0");
		if ($disk_info =~ /oneadmin oneadmin (\d*) /) {
			$bytes2 = $1;
		}
		verbose("$bytes <-----> $bytes2\n");
	}
	verbose("Successfully disk copy.....\n");
}

sub create_template_on_cloud1 {
	system("touch templates/template_$VM_ID.txt");
	my $tempfile = "templates/template_$VM_ID.txt";

	open(FILE, ">>$tempfile") or die "can't open '$tempfile': $!";
	my $i = 0;
	foreach (@mod_template) {
		print FILE "$_\n" unless $i == 0;
		$i++;
	}
	close(FILE);

	my $scp = Net::SCP::Expect->new;
	$scp->login($CLOUD1_USER, $CLOUD1_PW);
	$scp->scp($tempfile, "$CLOUD1_USER" . "@" . "$CLOUD1_IP:templates/");
}

sub create_template_on_cloud2 {
	system("touch templates/template_$VM_ID.txt");
	my $tempfile = "templates/template_$VM_ID.txt";

	open(FILE, ">>$tempfile") or die "can't open '$tempfile': $!";
	my $i = 0;
	foreach (@mod_template) {
		print FILE "$_\n" unless $i == 0;
		$i++;
	}
	close(FILE);

	verbose("Starts creating new template on cloud2.....\n");
	my $scp = Net::SCP::Expect->new;
	$scp->login($CLOUD2_USER, $CLOUD2_PW);
	$scp->scp($tempfile, "$CLOUD2_USER" . "@" . "$CLOUD2_IP:templates/");
	verbose("Successfully created new template on cloud2.....\n");
	sleep(1);
}
```

```
sub create_vm_on_cloud1 {
        verbose("Starts creating new VM on cloud1.....\n");
        $ssh_cloud1->exec("onevm create templates/template_$VM_ID.txt");
        verbose("Successfully created new VM on cloud1.....\n");
}

sub create_vm_on_cloud2 {
        verbose("Starts creating new VM on cloud2.....\n");
        $ssh_cloud2->exec("onevm create templates/template_$VM_ID.txt");
        verbose("Successfully created new VM on cloud2.....\n");
}

sub usage {
        # prints the correct use of this script
        print "Usage:\n";
        print "-s    Source IP\n";
        print "-D    Destination IP\n";
        print "-p    Source Password\n";
        print "-P    Destination Password\n";
        print "-u    Source User\n";
        print "-U    Destination User\n";
        print "-i    Virtual Machine ID from Source Cloud\n";
        print "-o    Path to Logfile\n";
        print "-h    Help\n";
        print "-v    Verbose\n";
        print "-d    Debug\n";;
}

sub verbose {
        print $_[0] if $VERBOSE;
}

sub debug {
        print "DEBUG: " . $_[0] if $DEBUG;
}
```

Appendix B

automation.pl

The automation.pl script

```
#! /usr/bin/perl -w

# Needed packages
use Getopt::Std;
use strict "vars";
use Net::SSH::Expect;
use POSIX qw(strftime);

# Global vars
my $VERBOSE = 0;
my $DEBUG = 0;

# Handle flags and args
# Example: c == "-c", c: == "-c argument"
my $opt_string = "vdhu:p:t:";
getopts("$opt_string",\my %opt ) or usage() and exit 1;

# Print help msg if -h is invoked
if ( $opt{h} ){
        usage();
        exit 0;
}
# Handle other user input
$VERBOSE = 1 if $opt{v};
$DEBUG = 1 if $opt{d};
my $CLOUD_IP = $opt{'c'};
my $CLOUD_USER = $opt{'u'};
my $CLOUD_PW = $opt{'p'};
my $DELAY = $opt{'t'};
my $LOGFILE = $opt{'o'};

(my $SOURCE_IP, my $DEST_IP) = split(/:/,$CLOUD_IP);
(my $SOURCE_USER, my $DEST_USER) = split(/:/,$CLOUD_USER);
(my $SOURCE_PW, my $DEST_PW) = split(/:/,$CLOUD_PW);


#print "Enter your password: \n";
#my $CLOUD1_PW = <>;

#---- Start: Main script content ----#

verbose("Verbose is enabled\n");
debug("Debug is enabled\n");

my $ssh_source = Net::SSH::Expect->new (
        host => $SOURCE_IP,
        password => $SOURCE_PW,
```

```
        user => $SOURCE_USER,
        raw_pty => 1,
        timeout => 3
);
$ssh_source->login();
verbose("Successfully logged into $SOURCE_IP.....\n");

open(FILE, ">$LOGFILE.tsv");
print FILE "";
close(FILE);

my $vms_tmp = $ssh_source->exec("onevm list $SOURCE_USER");
my @vms = split("\n",$vms_tmp);
my @ids;

foreach(@vms) {
        chomp $_;
        if ($_ =~ /^\s+(\d*)/) {
                push(@ids, $1);
        }
}

my $num_of_vms = scalar(@ids);

for (my $i = 0; $i < scalar(@ids); $i++) {
        my $local_time = strftime "%H:%M:%S", localtime;
        out("$local_time\t");
        out("$num_of_vms\n");

        print("=========== VM: $ids[$i] STARTS MIGRATING ===========\n");
        system("./manager.pl -v -s $SOURCE_IP -D $DEST_IP -p $SOURCE_PW -P $DEST_PW \
                                -u $SOURCE_USER -U $DEST_USER -i $ids[$i]");
        print("=========== VM: $ids[$i] IS MIGRATED ===========\n\n");

        $num_of_vms = $num_of_vms - 1;

        if ($num_of_vms >= 1) {
                print("========= NEXT VM WILL BE MIGRATED IN $DELAY SECONDS =========\n\n");
                sleep($DELAY);
        }
}

#---- End: Main script content ----#

sub out {
        open(OUT, ">>$LOGFILE") or die "Can't write to file '$LOGFILE' [$!]\n";
        print OUT $_[0];
        close(OUT);
}


sub usage {
        # prints the correct use of this script
        print "Usage:\n";
        print "-c source:destination    Cloud IP addresses\n";
        print "-u source:destination    Cloud user names\n";
        print "-p source:destination    Cloud passwords\n";;
        print "-t                      Delay time, in seconds, between each VM migration\n";
        print "-o                       Path to Logfile\n";
        print "-h                       Help\n";
        print "-v                       Verbose\n";
        print "-d                       Debug\n\n";

        print "./script -c ip:ip -u user:user -p pw:pw -t seconds [-o] [-d] [-v] [-h]\n\n";
}

```

```
114 sub verbose {
115         print "VERBOSE: " . $_[0] if $VERBOSE;
116 }
117
118 sub debug {
119         print "DEBUG: " . $_[0] if $DEBUG;
120 }
```

Appendix C

dest-log.pl

The dest-log.pl script

```
#! /usr/bin/perl -w

# Needed packages
use Getopt::Std;
use strict "vars";
use Net::SSH::Expect;
use POSIX qw(strftime);

# Global vars
my $VERBOSE = 0;
my $DEBUG = 0;

# Handle flags and args
# Example: c == "-c", c: == "-c argument"
my $opt_string = "vdhc:";
getopts("$opt_string",\my %opt ) or usage() and exit 1;

# Print help msg if -h is invoked
if ( $opt{h} ){
        usage();
        exit 0;
}

my $CLOUD_IP = $opt{'c'};

# Handle other user input
$VERBOSE = 1 if $opt{v};
$DEBUG = 1 if $opt{d};

#print "Enter your password: \n";
#my $CLOUD1_PW = <>;

#---- Start: Main script content ----#

verbose("Verbose is enabled\n");
debug("Debug is enabled\n");

my $ssh_dest = Net::SSH::Expect->new (
        host => $CLOUD_IP,
        password => [pw],
        user => [user],
        raw_pty => 1,
        timeout => 3
);
$ssh_dest->login();
verbose("Successfully logged in.....\n");

```

```
48  open(FILE, ">log10.tsv");
49  print FILE "";
50  close(FILE);
51
52  my $counter = 0;
53
54  while (1) {
55      my $vms_tmp = $ssh_dest->exec("onevm list oneadmin");
56      my @vms = split("\n",$vms_tmp);
57      my @runns;
58
59      my $local_time = strftime "%H:%M:%S", localtime;
60      out("$local_time\t");
61      verbose("$local_time\t");
62
63      foreach(@vms) {
64          chomp $_;
65          if ($_ =~ /(runn)/) {
66              $counter++;
67          }
68      }
69      out("$counter\n");
70      verbose("$counter\n");
71      $counter = 0;
72  }
73
74  sub out {
75      open(OUT, ">>log10.tsv") or die "Can't write to file 'log10.tsv' [$!]\n";
76      print OUT $_[0];
77      close(OUT);
78  }
79
80  sub usage {
81      # prints the correct use of this script
82      print "Usage:\n";
83      print "-c      Cloud IP address\n";
84      print "-h   Help\n";
85      print "-v   Verbose\n";
86      print "-d   Debug\n\n";
87  }
88
89  sub verbose {
90      print "" . $_[0] if $VERBOSE;
91  }
92
93  sub debug {
94      print "DEBUG: " . $_[0] if $DEBUG;
95  }
```

Appendix D

parallel.pl

The parallel.pl script

```
#! /usr/bin/perl -w

# Needed packages
use Getopt::Std;
use strict "vars";
use Net::SSH::Expect;

# Global vars
my $VERBOSE = 0;
my $DEBUG = 0;

# Handle flags and args
# Example: c == "-c", c: == "-c argument"
my $opt_string = "vdh";
getopts("$opt_string",\my %opt ) or usage() and exit 1;

# Print help msg if -h is invoked
if ( $opt{h} ){
        usage();
        exit 0;
}
# Handle other user input
$VERBOSE = 1 if $opt{v};
$DEBUG = 1 if $opt{d};

#---- Start: Main script content ----#

verbose("Verbose is enabled\n");
debug("Debug is enabled\n");

open(CMD1,"./manager.pl -v -s 128.39.74.2 -D 128.39.74.29 -p Hauk2.Lace -P Hauk2.Lace \
                        -u oneadmin -U oneadmin -i 1 -o logfile.txt & |");
open(CMD2,"./manager.pl -v -s 128.39.74.2 -D 128.39.74.29 -p Hauk2.Lace -P Hauk2.Lace \
                        -u oneadmin -U oneadmin -i 2 -o logfile.txt & |");
open(CMD3,"./manager.pl -v -s 128.39.74.2 -D 128.39.74.29 -p Hauk2.Lace -P Hauk2.Lace \
                        -u oneadmin -U oneadmin -i 3 -o logfile.txt & |");
open(CMD4,"./manager.pl -v -s 128.39.74.2 -D 128.39.74.29 -p Hauk2.Lace -P Hauk2.Lace \
                        -u oneadmin -U oneadmin -i 4 -o logfile.txt & |");


#---- End: Main script content ----#
```

```
48  sub usage {
49      # prints the correct use of this script
50      print "Usage:\n";
51      print "-h    Help\n";
52      print "-v    Verbose\n";
53      print "-d    Debug\n";;
54
55      print "./script    [-d] [-v] [-h]\n";
56  }
57
58  sub verbose {
59      print "VERBOSE: " . $_[0] if $VERBOSE;
60  }
61
62  sub debug {
63      print "DEBUG: " . $_[0] if $DEBUG;
64  }
```

Appendix E

log-parser.pl

The logparser.pl script

```
#! /usr/bin/perl -w

# Needed packages
use Getopt::Std;
use strict "vars";
use Net::SSH::Expect;

# Global vars
my $VERBOSE = 0;
my $DEBUG = 0;

# Handle flags and args
# Example: c == "-c", c: == "-c argument"
my $opt_string = "vdh";
getopts("$opt_string",\my %opt ) or usage() and exit 1;

# Print help msg if -h is invoked
if ( $opt{h} ){
        usage();
        exit 0;
}
# Handle other user input
$VERBOSE = 1 if $opt{v};
$DEBUG = 1 if $opt{d};

#---- Start: Main script content ----#

verbose("Verbose is enabled\n");
debug("Debug is enabled\n");

open(LOG, "logfile.txt");
open(OUT, ">>out.txt");

my $sec = 0;

foreach(<LOG>) {
        if ($_ =~ /^(\d*)\s*(\d*)/) {
                $sec = $2 - $1;
                print OUT "$sec\n";
        }
}

close(LOG);
close(OUT);
```

```
48  sub usage {
49      # prints the correct use of this script
50      print "Usage:\n";
51      print "-h    Help\n";
52      print "-v    Verbose\n";
53      print "-d    Debug\n";
54
55      print "./script  [-d] [-v] [-h]\n";
56  }
57
58  sub verbose {
59      print "VERBOSE: " . $_[0] if $VERBOSE;
60  }
61
62  sub debug {
63      print "DEBUG: " . $_[0] if $DEBUG;
64  }
```

Appendix F

disk-load.pl

The disk-load.pl script

```
#! /usr/bin/perl -w

# Needed packages
use Getopt::Std;
use strict "vars";
use Net::SSH::Expect;

# Global vars
my $VERBOSE = 0;
my $DEBUG = 0;

# Handle flags and args
# Example: c == "-c", c: == "-c argument"
my $opt_string = "vdhs:";
getopts("$opt_string",\my %opt ) or usage() and exit 1;

# Print help msg if -h is invoked
if ( $opt{h} ){
        usage();
        exit 0;
}
# Handle other user input
$VERBOSE = 1 if $opt{v};
$DEBUG = 1 if $opt{d};

#---- Start: Main script content ----#

verbose("Verbose is enabled\n");
debug("Debug is enabled\n");

system("touch load");

my $filesize = $opt{s};
system("dd if=/dev/zero of=./load bs=1024 count=$filesize");

sub usage {
        # prints the correct use of this script
        print "Usage:\n";
        print "-h    Help\n";
        print "-v    Verbose\n";
        print "-d    Debug\n";
        print "-s    Filesize in bytes\n";

        print "./script   -s [Filesize] [-d] [-v] [-h]\n";
}


```

```
48  sub verbose {
49      print "VERBOSE: " . $_[0] if $VERBOSE;
50  }
51
52  sub debug {
53      print "DEBUG: " . $_[0] if $DEBUG;
54  }
```